3DCubic

Visualisierung von Motion Capture-Datenformaten mit Macromedia Director

Diplomarbeit

Studiengang Audiovisuelle Medien
der
Fachhochschule Stuttgart –
Hochschule der Medien

Cosima Schunk

Erstprüfer: Prof. Dr. Thomas Keppler
Zweitprüfer: Prof. Dr. Bernhard Eberhardt

Bearbeitungszeitraum: 15. Juni 2004 bis 14. Oktober 2004

Stuttgart, Oktober 2004

Kurzfassung

Diese Diplomarbeit hat das Erstellen einer Anwendung zum Ziel, die es ermöglicht, unterschiedliche Dateiformate, die zur Speicherung von Motion- Capture-Daten verwendet werden, betrachten zu können. Zur Realisation dieses „MoCap-Viewers“ wird Macromedia Director verwendet, das seit der Version 8.5 über die Möglichkeit der Einbindung von Echtzeit-3D-Grafik verfügt.

Die Anwendung soll die Möglichkeit bieten, einzelne Ordner oder auch komplette Festplatten nach ausgewählten Formaten (*.csm, *.c3s und/oder *.amc/*.asf) zu durchsucht, wichtige Informationen aus diesen Dateien auszulesen und schließlich auch mit Hilfe von Shockwave 3D die Marker- beziehungsweise Rotationsdaten im 3D-Raum darzustellen. Darüber hinaus wird überprüft, ob ein zugehöriges Informationsfile (*.i3d) vorliegt. Falls nicht, wird die Erstellung dieser Datei angeboten, wobei auch die aus der entsprechenden 3D-Datei bekannten Daten eingetragen werden. Des Weiteren können bereits bestehende Informationsdateien editiert werden.

Die Anwendung soll es Studenten ermöglichen, Motion Capture-Files schnell durchzusehen, ohne auf teure Spezialsoftware zurückgreifen zu müssen, da für diese oft nur eine begrenzte Anzahl an Lizenzen und somit Arbeitsplätzen zur Verfügung steht. Außerdem soll ihnen die Möglichkeit geboten werden, beim Durchsehen auch gleich noch Informationen, zum Beispiel über besonders gut verwendbare Segmente, in den i3d-Files festzuhalten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

3D	dreidimensional
3DPI	3D Property Inspector
amc	Acclaim Motion Capture Data
asf	Acclaim Skeleton File
c3d	Coordinate 3D
csm	Character Studio 2.0 Motion Capture File Format
dof	degree-of-freedom
GUID	Globally Unique Identifier
HdM	Hochschule der Medien
MLS	Motion Lab System
MOA	Macromedia Open Architecture
XDK	XTRA Development Kit

1 Einleitung

1.1 Stand der Technik

Motion Capture ist in der Computeranimation mittlerweile eine weit verbreitete Technik, um realistische Bewegungsinformationen zu erhalten. Auch die Fachhochschule Stuttgart, Hochschule der Medien (HdM) verfügt über ein optisches Motion Capture-System von *Vicon Motion Systems Ltd.*

Im Laufe der Zeit haben sich verschiedene Formate etabliert, in denen die gewonnenen optischen Daten gespeichert werden können. Leider ist es oft nicht möglich, diese Dateien ohne weiteres einzusehen. Im Falle der von Vicon genutzten c3d-Files ist es noch nicht einmal möglich, sich auch nur die gespeicherten Zahlenwerte der Markerkoordinaten im 3D-Raum anzusehen, da sie als Binärdatei gespeichert werden. Gängige c3d-File-Editoren wie zum Beispiel der C3dEditor von *Motion Lab System* (MLS) ermöglichen es zwar, die Inhalte der Datei auszulesen, die grafische Darstellung der Informationen beschränkt sich aber auf das Zweidimensionale.

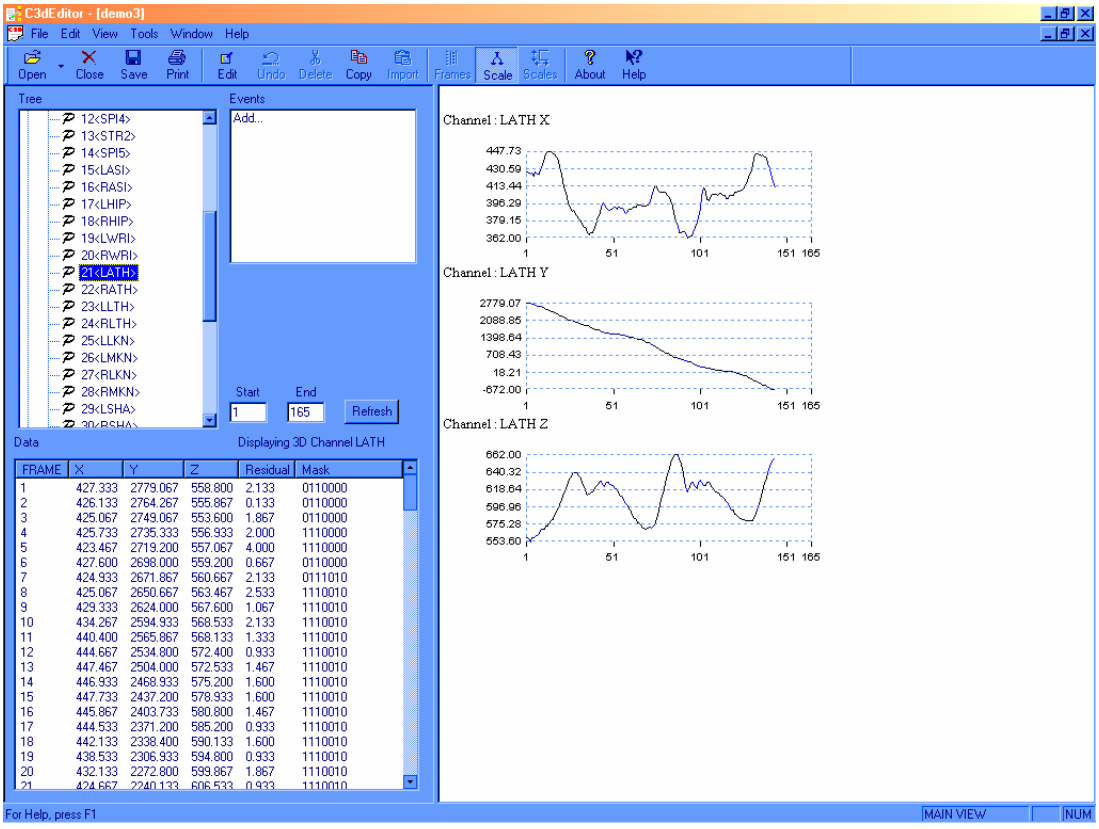

Abbildung 1: MLS C3dEditor, Darstellung von 3D-Markerdaten

Bei den anderen in dieser Arbeit berücksichtigten Formaten, dem Acclaim-Format (asf/amc) und dem von *Discreets* Character Studio genutzten csm-Format können zwar die Inhalte der Dateien direkt angesehen werde, da es sich um ASCII-Dateien handelt, aber auch hier ist eine Betrachtung der Zahlenkolonnen, in denen die Bewegungsdaten festgehalten sind, wenig hilfreich.

Selbstverständlich sind viele Programme auf dem Markt, die die dreidimensionale Darstellung dieser Dateiinhalte ermöglichen, wie zum Beispiel Motionbuilder von *Kaydara Inc.*, BodyBuilder von *Vicon*, aber diese Programme sind für die professionelle Weiterverarbeitung gedacht und als reines Betrachtungstool ganz einfach „überqualifiziert“

1.2 Ziele

Ziel dieser Arbeit ist es, der HdM, insbesondere den Studenten und Mitarbeitern, die sich mit Computeranimation und Motion Capture befassen, die Möglichkeit zu bieten, Motion Capture-Daten in einigen der gängigen Formaten einfach sichten zu können, ohne dafür auf teure Spezialsoftware zurückgreifen zu müssen. Die ausgewählten Formate sind das c3d-Format, das von der Vicon Workstation zur Speicherung der Markerdaten genutzt wird, das csm-Format, das vor allem im Characterstudio Verwendung findet und schließlich das Acclaim-Format, bestehend aus asf- und acm-Dateien, das keine Markerrohdaten mehr enthält, sondern die Beschreibung einer Skeletthierarchie und die Rotationsdaten der einzelnen Gelenke.

Dem Anwender soll es ermöglicht werden, die Festplatten seines PCs oder auch nur einzelne Ordner und Unterorder nach vorhandenen Dateien zu durchsuchen, sich das Suchergebnis auflisten und alle zugehörenden Informationen anzeigen zu lassen.

Des Weiteren soll die Möglichkeit zur Verfügung gestellt werden, Informationsdateien zu erstellen, in denen die wichtigsten Informationen zu den einzelnen Dateien abgelegt werden können und die für alle oben genannten Dateiformate das selbe Aussehen haben.

2 Das c3d-Fileformat

Alle Tabellen dieses Kapitels sind dem „C3D Format User Guide“ von *Motion Lab System* (MLS) entnommen.

2.1 Überblick

Das Coordinate 3D-Fileformat (c3d) ist Teil einer ganzen Familie von Fileformaten, die ursprünglich für das AMASS Fotogrammetrie Software System entwickelt wurde.

Das c3d-Format bietet die Möglichkeit, sowohl 3D-Koordinaten, als auch analoge Daten, so wie alle weiteren zugehörigen Parameter eines einzelnen Messvorgangs innerhalb einer Datei abzulegen.

Seine weiteste Verbreitung hat das c3d-Format im Bereich der klinischen Ganganalyse und in biomechanischen Instituten. Es ist weltweit das wahrscheinlich am meisten verwendete Dateiformat für biomechanische 3D-Datenerfassung. Aber es wird auch in einigen anderen Gebieten gerne verwendet, wie zum Beispiel in der Unterhaltungs- und Animationsindustrie.

Einige wichtige Kriterien bei der Entwicklung des c3d-Formates waren:

- flexible Speicherung verschiedenere Datenarten innerhalb einer Datei
- flexible Speicherung von Parametern und Parameterarten innerhalb einer „Parametersektion“
- Abspeicherung von Parametern mit aussagekräftigen Namen und Beschreibungstexten, so dass die Datei ihren Inhalt selbst erklären kann
- die Möglichkeit für Anwender, eigene Parameter in schon bestehende Dateien einzufügen oder sie ergänzen zu können
- eine effiziente und kompakte Speicherung aller benötigten Daten innerhalb einer einzigen Datei

2.2 Kurzere Entstehungsgeschichte

Beginn der 1980er Jahre:
SEALSPOT System verwendet einen Vorläufer des c3d-Fromates.

Ende der 80er:
Oxford Metrics Ltd. übernimmt die Vermarktungsrechte der AMASS Software von *ADTech*. Zu diesem Zeitpunkt wird bereits das c3d-Format als Ausgabeformat verwendet.

1991:
Dwight Meglan veröffentlicht ANZ im Rahmen seiner Doktorarbeit an der Ohio State University. ANZ ist ein Bewegungsanalyse-Paket, das unter MS-DOS läuft.

Die frühen 90er:
AMASS wird angepasst, um Videorohdaten von den Systemen anderer Anbieter verarbeiten zu können, und wird als Drittanbietersoftware für etliche Motion Capture-System angeboten.

1992:
Vicons Clinical Manager wird unter Windows 3.1 eingeführt. Diese Anwendung ermöglicht die schnelle Darstellung klinischer Daten aus Motion Capture-Daten. Ihre Popularität sorgt dafür, dass das c3d-File-Format zu seiner heutigen Verbreitung kommt.

Mitte der 90er:
Wegen des großen Erfolges von Vicon-VX, entwickelt Oxford Metrics eine neue Hardware-Plattform, die auf dem Betriebssystem Windows basiert und eine eigene fotogrammetrische Software verwendete Dieses Paket ersetzt die über die Kommandozeile gesteuerte AMASS Software und ist das erste professionelle Fotogrammetrie-Paket auf dem Markt, das unter Windows läuft. Auch dieses Paket verwendet das ADTech c3d-Format als Standardausgabeformat.
Ungefähr zur selben Zeit überträgt ADTech die AMASS Software von DEC-Rechnern auf Intel PCs und erweitert das c3d-Format um zu ermöglichen, dass es auf den verschiedenen Computersystemen verwendet werden kann.
Außerdem wird in dieser Zeit MOVE3D auf den Markt gebracht. Ein hoch entwickeltes Analyseprogramm für 3D-Daten, das von Tom Kepple beim NIH Clinical Center, National Institutes of Health entwickelt worden ist und c3d-Dateien als Eingabeformat nutzt.

1997:
Neben den Softwareangeboten von Oxford Metrics und ADTech wird der C3dEditor von Motion Lab Systems die erste wichtige kommerzielle c3d-

Anwendung. Sie ermöglicht dem Anwender zum ersten Mal die einfache Manipulation von c3d-Daten in der grafischen Windowsumgebung.

Ende der 90er gibt es fünf große Firmen, die unabhängige c3d-Anwendungen im Angebot haben: ADTech, Oxford Mertrics, Motion Analysis Corporation, Motion Lab Systems und ANZ.

2.3 Aufbau des c3d-Files

Jedes c3d-File besteht im Wesentlichen aus drei Teilen:

- der Headersektion
- der Parametersektion
- der Datensektion

Insgesamt ist das c3d-File aus einer Summe von 512-Byte Blöcken zusammengesetzt.

2.3.1 Die Headersektion

Der Header umfasst den ersten 512-Byte-Block des c3d-Files. Er enthält neben einigen anderen Daten auch Kopien von Inhalten des Parameterblocks.
Er besteht aus 256 16-Bit-Worten, die in folgender Struktur dargestellt werden:

Tabelle 1: Die 256 Worte der c3d-Headersektion

16-Bit-Wort	Beschreibung
Wort 1	in zwei 8-Bit-Worte aufgeteilt (2x unsigned char) 1. Wort: Nummer des 512-Byte-Blocks, in dem die Parametersektion anfängt 2. Wort: Hexadezimalwert 0x50 (dezimal 80) als Kennzahl, dass die Datei wirklich im c3d Formt abgespeichert ist
Wort 2	Short Integer, Anzahl der 3D-Punkte/Trajektorien
Wort 3	Short Integer, Anzahl der Analogkanäle/ 3D-Frame
Wort 4	Short Integer, Nummer des ersten Frames, Zählung beginnt mit 1, nicht mit
Wort 5	Short Integer: Nummer des letzten Frames
Wort 6	Short Integer: Maximale Interpolationslücke der 3D-Frames
Wort 7/8	32-Bit-Fließkommazahl: 3D-Skalierungsfaktor, zum Umrechnen der 3D-Daten in Systemeinheiten, falls diese als Integer abgelegt wurden. Ist diese Zahl negativ, wurden die Daten als Fließkommazahlen abgespeichert

Wort 9	Nummer des 512-Byte-Blocks, in dem die Datensektion beginnt
Wort 10	Short Integer, Anzahl analoger Abtastwerte pro 3D-Frame
Wort 11/12	32-Bit-Fließkommazahl, Abtastrate in Hertz
Worte 13 – 147	Reserviert für zukünftige Nutzung
Wort 148	Short Integer, 0x3039, bzw. 12345_{10} , Indikator, ob Label- und Range-Daten vorliegen
Wort 149	Short Integer, erster Block der Label- und Range-Sektion – falls diese vorhanden ist
Wort 150	Short Integer, 0x3039, bzw. 12345_{10} , Indikator, ob Labels unterstützt werden, deren Bezeichnung vier Zeichen (char) umfasst
Wort 151	Short Integer, Anzahl der definierten Zeitereignisse (1 bis 18)
Wort 152	Reserviert für zukünftige Nutzung
Worte 15 – 188	Fließkommazahlen, Ereigniszeiten in Sekunden
Worte 189 – 197	Short Integers, 18 Flags für die Ereignisdarstellung
Wort 198	Reserviert für zukünftige Nutzung
Worte 199 – 234	8-Bit-Characters, Ereignislabels, jedes Label besteht aus vier Characters
Worte 235 – 256	Reserviert für zukünftige Nutzung

2.3.2 Die Parametersektion

Die Parametersektion besteht aus einem oder mehreren 512-Byte-Blöcken. Sie enthält zusätzliche Informationen über die 3D-Daten und/oder Analogdaten der Datei.

Innerhalb der Parametersektion werden jedem abgespeicherten Parameter ein Name und ein Datentyp zugeordnet. Einige Parameter sind zwingend vorgeschrieben, andere können vom Anwender frei hinzugefügt werden. Zusammengehörende Parameter werden zu Gruppen zusammengefasst. Jeder Parameter der Datei ist einer dieser Gruppen zugeordnet. Jede Gruppe hat einen einmaligen Namen und kann eine Gruppenbeschreibung enthalten.

Es gibt vier verschiedene Datentypen, die ein Parameter enthalten kann: (Short) Integer, Fließkommazahlen, Characters oder Bytes (signed 8-Bit Zahlen).

Die Parametersektion hat folgende Struktur:

Der Parameterheader besteht aus vier Bytes:

Tabelle 2: Der Header der c3d-Parametersektion

Byte	Beschreibung
1	0x00, reserviert
2	0x00, reserviert
3	Anzahl der noch folgenden 512-Byte-Blöcke der Parametersektion
4	dezimal 83 + Identifikationsnummer des Prozessortyps Prozessortyp 1 = Intel Prozessortyp 2 = DEC (VAX, PDP-11) Prozessortyp 3 = MIPS processor (SGI)

Die eigentlichen Parameterdaten beginnen mit dem fünften Byte des ersten Blocks der Parametersektion.

Die gespeicherten Parameter sind in Gruppen organisiert. Jeder Parametername innerhalb einer Gruppe muss einmalig sein. Aber sie können durchaus auch ein zweites Mal verwendet werden, wenn dies in einer anderen Gruppe geschieht, beispielsweise gehört der Parametername RATE sowohl zur Gruppe POINT als auch zur Gruppe RATE. Jeder Parameter ist über eine ID seiner Gruppe zugeordnet, so dass es zu keiner Verwirrung kommen kann, welches RATE wann gemeint ist.

Innerhalb der Parametersektion werden also zwei unterschiedliche Formate abgelegt: Das Gruppenformat und das Parameterformat. Diese beiden Formate können in willkürlicher Reihenfolge hintereinander geschrieben werden.

Tabelle 3: Das Gruppenformat der c3d-Parametersektion

Byte	Länge (Byte)	Beschreibung
1	1	Anzahl der Zeichen des Gruppennamens. Falls dieser Wert negativ ist, ist das das Zeichen dafür, dass die Gruppe „gesperrt“ ist.
2	1	Gruppen ID (-1 bis -127, immer negativ!)
3	n	Gruppenname (ausschließlich ASCII-Zeichen – Großbuchstaben A - Z, 0 - 9 und underscore _)
3+n	2	Signed Integer, gibt in Bytes den Versatz zum Start der nächsten Sektion (Gruppe oder Parameter) an
3+n+2	1	Anzahl der Zeichen in der Gruppenbeschreibung
3+n+3	m	Gruppenbeschreibung (ASCII-Zeichen, Groß- und Kleinbuchstaben gemischt)

Tabelle 4: Das Parameterformat der c3d-Parameterektion

Byte	Länge (Byte)	Beschreibung
1	1	Anzahl der Zeichen des Parameternamens – wenn der Wert negativ ist, ist der Parameter „gesperrt“
2	1	ID der Gruppe (positiver Wert) zu welcher der Parameter gehört
3	n	Parametername (ausschließlich ASCII-Zeichen – Großbuchstaben A - Z, 0 - 9 und underscore _)
3+n	2	Signed Integer, gibt in Bytes den Versatz zum Start der nächsten Sektion (Gruppe oder Parameter) an
3+n+2	1	Länge in Byte der Datenelemente, die dieser Parameter enthält: -1 Character (1 Byte) 1 Bytedaten (1 Byte) 2 Integerdaten (2 Byte) 4 Floatingpointdaten (4 Byte)
3+n+3	1	Anzahl der Dimensionen des Parameters (0-7) – 0, falls der Vektor nur eine Dimension hat
3+n+4	d	Parameterdimensionen
3+n+4+d	t	Parameterdaten
3+n+4+d+t	1	Anzahl der Zeichen in der Parameterbeschreibung
3+n+4+d+t+1	m	Parameterbeschreibung (ASCII-Zeichen, Groß- und Kleinbuchstaben gemischt)

Der Sperrmechanismus für Gruppen und Parameter war ursprünglich dafür vorgesehen, Nutzer von Editierprogrammen wie PRM oder C3dEditor daran zu hindern, Gruppen und Parameter zu verändern. Seine Effektivität hängt davon ab, in wie weit das genutzte Programm auf diesen Faktor eingeht. In der Realität stellt das Sperren von Gruppen lediglich eine Marke da ob die Gruppe/der Parameter geändert werden darf oder nicht. Sie verhindert nicht, dass eine Gruppe/ein Parameter, der in einer Anwendung gesperrt wurde, in einer anderen geändert werden kann.

Die Parametersektion wird korrekterweise mit 0 abgeschlossen, was so viel bedeutet wie: Diese Gruppen-/Parametername hat die Länge 0.

Die Anzahl der Parameterdimensionen kann zwischen 0 und 7 variieren. Ein Parameter mit der Dimension 0 ist skalar. Ist der Parameter eine Matrix, so sind die tatsächlichen Dimensionsgrößen im folgenden Byte gespeichert (zum Beispiel 2x3, 6x6 und so weiter) danach folgen die Daten selbst.

2.3.3 Die Datensektion

Die 3D-Punktdaten und die analogen Abtastwerte werden einfach ab Beginn des ersten 512-byte-Blocks der Datensektion hintereinander geschrieben. Enthält die

Datensektion sowohl 3D-, als auch Analogdaten, so werden zuerst die Frames der 3D-Daten geschrieben, dann erst die Analogwerte.

Tabelle 5: Reihenfolge der Daten in der c3d-Datensektion

3D-Daten von Frame 1
Analogdaten von Frame 1
3D-Daten von Frame 2
Analogdaten von Frame 2
…
3D-Daten von Frame n (n = Anzahl Frames, siehe Header Wort 5 – Wort 3+1)
Analogdaten von Frame n

Es gibt keine end-of-data-Marke. Alle Werte werden vom ersten bis zum letzten Frame hintereinander geschrieben, so dass beim Auslesen einfach mitgezählt werden kann.

Die 3D- und die Analogwerte werden entweder als Integer oder als Fließkommazahlen abgelegt. Welches Format auch immer gewählt wurde, es muss zwingend sowohl für die 3D- als auch für die Analogwerte dasselbe Format verwendet werden. Es ist nicht möglich, beide Formate zu mischen, da es nur eine einzige Marke (Wort 7/8 der Headersektion) gibt, an der abgelesen werden kann, welches Format verwendet wurde.

Die Reihenfolge, in der die 3D-Markerdaten in der Datensektion abgelegt sind, ist dieselbe, wie die Reihenfolge (einer der Parameter, die zwingend erforderlich sind), die in der Parametersektion beschrieben wurde.

Tabelle 6: 3D-Daten im Integerformat in der c3d-Datensektion

Wort	Inhalt
1	X-Koordinate des Markers geteilt durch den POINT:SCALE Faktor
2	Y-Koordinate des Markers geteilt durch den POINT:SCALE Faktor
3	Z-Koordinate des Markers geteilt durch den POINT:SCALE Faktor
4	Byte 1: Maske, die angibt, welche Kameras den Marker gesehen haben Byte 2: Durchschnittliche Abweichung geteilt durch POINT:SCALE Faktor

Tabelle 7: 3D-Daten im Floatformat in der c3d-Datensektion

Wort	Inhalt
1 - 2	X-Koordinate des Markers
3 - 4	Y-Koordinate des Markers
5 - 6	Z-Koordinate des Markers

7 - 8	Nach der Konvertierung in signed Integer: Byte 1: Maske, die angibt, welche Kameras den Marker gesehen haben Byte 2: Durchschnittliche Abweichung geteilt durch POINT:SCALE Faktor

3 Das csm-Fileformat

3.1 Überblick

Das csm-Fileformat ist ein ASCII-Fileformat, in dem ebenfalls Informationen zu optischen Motion Capture-Markerdaten abgespeichert werden. Neben den Markerbezeichnungen, ihren Labeln, sind einzelbildeweise die X-, Y- und Z-Koordinaten der einzelnen Marker gespeichert.

Zusätzlich ist es möglich, einige weitere Informationen, wie beispielsweise das Aufnahmedatum, oder der Name des Darstellers speicherbar.

3.2 Kurze Entstehungsgeschichte

Das Character Studio 2.0 Motion Capture- Filmformat (csm) beschreibt den Aufbau einer ASCII-Datei, die dazu entwickelt wurde, den Austausch zwischen Motion Capture-Systemen und descreet Character Studio zu ermöglichen.

3.3 Aufbau des csm-Files

Der Aufbau der csm-Datei ist sehr einfach. Es gibt eine gewisse Anzahl optionaler Informationen und zwei Abschnitte mit Pflichtinformationen.

Jeder Informationsunterpunkt startet mit einem Dollarzeichen ‚$' gefolgt von einem Namen, der wahlweise in Groß- oder Kleinbuchstaben geschrieben sein kann. Die zugehörigen Daten können sowohl in derselben Zeile wie der Namen als auch in der Folgezeile stehen.

Tabelle 8: Mögliche Unterpunkte einer csm-Datei

Name	Format
$comments	String
$firstframe	Integer
$lastframe	Integer
$actor	String
$rate	Integer

Die beiden Informationsteile, die in jedem Fall vorliegen müssen sind zum einen $order und zum anderen $points.

$order enthält hintereinander geschrieben die Markernamen, in der Reihenfolge, in welcher dann die Koordinaten in der $point-Sektion abgelegt sind. Die Markernamen müssen nicht zwingend den unterstützten Markernamen des Character Studios entsprechen. Sie dürfen von beliebiger Länge sein, jedoch kein Kommas enthalten.

In dem Abschnitt, der mit dem Signalwort $points eingeleitet wird, stehen die Marker-Positionskoordinaten. Jedes Einzelbild nimmt eine eigene Zeile ein. Jede Zeile startet mit einem Integerwert, der die Framenummer angibt. Danach stehen als Fließkommazahlen für jeden Marker die X-, Y- und Z-Koordinaten. Es wird automatisch davon ausgegangen, dass für jeden Marker drei Werte abgespeichert wurden. Wenn dies nicht der Fall ist, weil ein so genannter „Dropout“ aufgetreten ist, das heißt ein Marker nicht von ausreichend Kameras gesehen werden konnte, um seine Position im Raum errechnen zu können, so muss dies kenntlich gemacht werden. Das Ende dieses Abschnitts ist entweder mit dem Ende der Datei erreicht, oder mit dem Auftreten eines weiteren Dollarzeichens.

Alle Angaben sind grundsätzlich in Millimetern und es wird davon ausgegangen, dass die nach oben weisende Achse die Z-Achse ist.

```
$Filename test.csm
$Date 2003/10/17
$Time 09:12:04
$Actor name


$FirstFrame 35
$LastFrame 732
$Rate 120.000

$order
LFHD RFHD LBHD RBHD C7 T10 CLAV STRN RBAC LSHO LELB LFRM LWRA LWRB LFIN RSHO RELB RFRM RWRA
RWRB RFIN LFWT RFWT LBWT RBWT LKNE LSHN LANK LHEE LTOE LMT5 RKNE RSHN RANK RHEE RTOE RMT5

$Points
35 210.340815 -1172.764274 1638.471262 360.408230 -1187.396232 1618.602603 243.403906 -
1342.751688 1569.110647 356.763076 -1305.067979 1577.735801 277.853183 -1334.075194
1424.844674 240.426174 -1369.140553 1288.331072 270.819575 -1156.489429 1341.160142
273.181224 -1116.495411 1231.908189 358.097921 -1372.374986 1332.380967 69.103914 -
1252.084889 1407.440345 9.754639 -1270.156640 1109.307782 -17.044948 -1262.917672
933.570265 53.907214 -1160.904687 864.209649 -30.239380 -1256.962208 854.609031 3.953196 -
1191.400768 785.248416 439.831525 -1257.064888 1424.793334 525.107604 -1277.190248
1097.448195 547.337912 -1209.626540 918.424905 497.435234 -1088.669020 906.462637
568.592756 -1179.335820 859.126969 535.889047 -1087.282835 809.583673 157.563085 -
1156.797470 930.592533 403.123280 -1147.453553 937.985522 223.740608 -1361.798904
1045.491909 343.619984 -1362.928388 1005.549230 138.977932 -1158.697058 467.657916
171.835662 -1294.953959 298.954006 158.179168 -1482.910445 201.767001 DROPOUT 232.879164 -
1423.355808 67.563708 162.748446 -1445.124055 82.195666 421.965134 -1186.215407 493.738739
346.289674 -1155.924687 228.925968 363.694003 -1212.604272 86.610924 310.813593 -
1245.051281 48.773194 306.603697 -1048.264280 46.719585 373.037920 -1078.041598 36.810926
```

Abbildung 2: Auszug aus einem csm-File

4 Das Acclaim-Format

4.1 Überblick

Das Acclaim-Format, ist ebenfalls ein ASCII-File. Es besteht aus zwei getrennten Dateien: asf und amc.

Im Gegensatz zu dem c3d- und dem csm-Format werden hier nicht die Positionsdaten von Markern angegeben, sondern die Rotationsdaten von Knochen (englisch Bones) einer Skelettstruktur.

Die asf-Datei enthält alle Angaben, die zum Aufbau des Skeletts erforderlich sind. Im amc-File stehen lediglich die Rotationsdaten der einzelnen Bones.

4.2 Kurze Entstehungsgeschichte

Das Format ist nach der *Acclaim Entertainment Inc.* Benannt, einer amerikanischen Computerspielefirma, die sich seit vielen Jahren mit der Anwendung von Motion Capturing in der Spieleindustrie beschäftigt. Sie haben eine eigene Methode entwickelt, um die optischen Daten in Skelettbewegungsdaten umzusetzen, und das Acclaim-Skeleton-File (asf) zur Speicherung der Skelettstruktur und das Acclaim-Motion-Capture-Data-File (amc) zur Speicherung der Bewegungen definiert. Nach einiger Zeit wurde das Format von Acclaim für die allgemeine Nutzung freigegeben. *Oxford Metrics*, die Entwickler des Vicon Motion Capure-Systems übernahmen es als Ausgabeformat für ihre Software. So bietet zum Beispiel BodyBuilder die Möglichkeit des Exports von *.asf und *.amc-Dateien.

4.3 Aufbau des asf-Files

Jeder Unterpunkt innerhalb des asf-Files beginnt mit einem Doppelpunkt gefolgt von einem Schlüsselbegriff.

Die Header-Sektion der Datei besteht theoretisch aus sechs Schlüsselinformationen, von denen drei zwingend erforderlich, die anderen drei optional sind.

Auch Pflichtinformationen müssen nicht zwingend aufgeführt sein. Ihr Fehlen sagt aus, dass ein Defaultwert vorausgesetzt wird.

Tabelle 9: Die Schlüsselbegriffe der asf-Headersektion

Schlüsselbegriff	Inhalt	
:version	Die Versionsnummer des asf-Fileformats	Pflicht
:name	Max. 50 Zeichen, Möglichkeit, dem Skelett einen anderen Namen als der Datei zu geben	Pflicht
:axis-rotation	Wenn diese Angebe fehlt, sind die Rotationsachsen des Root und der anderen Bones global zu interpretieren	Pflicht
:skin		Optional
:units	Unterpunkte: mass: Skalierungsfaktor für die Umrechnung fileinterne Gewichtseinheiten in die benötigte Ausgabeeinheit length: Skalierungsfaktor für die Umrechnung fileinterner Längeneinheiten in die benötigte Ausgabeeinheit angle: gibt an ob die Winkelangaben in Grad (deg) oder radial (rad) vorliegen	Optional
:documentation		optional

Ein asf-Header sieht dann in etwa folgendermaßen aus:

```
# AST/ASF file generated using VICON BodyLanguage
# ----------------------------------------------
:version 1.10
:name VICON
:units
 mass 1.0
 length 25.4
 angle deg
:documentation
  .ast/.asf automatically generated from VICON data using
  VICON BodyBuilder and BodyLanguage model BRILLIANT.MOD
```

Abbildung 3: Beispiel für den Header einer asf-Datei

Der Header wird gefolgt von den benötigten Angaben des Root-Bones des Skeletts, dieser Abschnitt beginnt mit dem Schlüsselwort :root.

Die Rootsektion hat vier weitere Unterpunkte, die keine bestimmte Reihenfolge haben müssen:

Tabelle 10: Die Unterpunkte der asf-Rootsektion

Unterpunkt	Inhalt
order	Gibt die Reihenfolge an, in der im amc-File die Informationen über Translation und Rotation für jeweils X, Y und Z abgespeichert sind
position	Gibt die Ausgangsposition relativ zu den globalen Achsen an
axis	Gibt die Rotationsreihenfolge der Drehung der vordefinierten Rotationsachsen des Rootbones an. Ist dieser Wert auf 0 gesetzt (bei BodyBuilder immer der Fall), hat dieser Unterpunkt keine Auswirkung
orientation	Gibt die vordefinierte Ausrichtung der lokalen Achsen des Rootbones bezogen auf das Weltkoordinatensystem an

Innerhalb der Datei stellen sich die Eintragungen der :root-Sektion beispielsweise so dar:

```
:root
  order TX TY TZ RX RY RZ
  axis XYZ
  position 0 0 0
  orientation 0 0 0
```

Abbildung 4: Beispiel für die :root-Sektion einer asf-Datei

Als nächstes werden in der *:bonedata*-Sektion die einzelnen Bones des Skeletts definiert. Die Definition jedes einzelnen Bones beginnt mit dem Schlüsselwort *begin* und endet wenn das Schlüsselwort *end* erreicht wird. Auch die Bonedefinition verfügt über mehrere Unterpunkte:

Tabelle 11: Die Unterpunkte der asf-Bonedata-Sektion

Unterpunkt	Beispiel	Inhalt
id	id 1	Optional: Kennziffer des Knochens
name	name xyz	Der Name des Knochens, er muss exakt mit dem Namen übereinstimmen, der auch um zugehörigen csm-File verwendet wurde
direction	direction 0.00 0.00 1.00	Vektor, der die Richtung des Childknochens in Bezug auf seinen Elternknochen angibt.
length	length 1.00	Die Länge des Vectors, dessen Richtung unter direction definiert wurde
bodymass	bodymass 1.0	Opotinal
cofmass	cofmass 1.0	Optional
axis	axis 0.00 20.00 0.00 XYZ	Gibt die lokalen Rotationsachsen des Knochens bezüglich des Weltkoordinatensystems an und die Rotationsreihenfolge
dof	tx ty rx ry rz l	degrees-of-freedom Gibt an, ob und welche Freiheitsgrade bezüglich des Elternknochens für diesen Knochen im zugehörigen amc-File gespeichert wurden
limits	limits 0.00 0.00 0.00 1.00	Optional, gibt Grenzwerte an, die die oben aufgeführten dofs nicht überschreiten dürfen

Eine typische Bonebeschreibung hat in etwa folgendes Aussehen:

```
begin
  id 2
  name lfemur
  direction 0.34202 -0.939693 0
  length 446.016
  axis 0 0 20 XYZ
 dof rx ry rz
 limits (-160.0 20.0)
     (-70.0 70.0)
     (-60.0 70.0)
end
```

Abbildung 5: Auszug aus einer :bonedata-Sektion einer asf-Datei

Den Abschluss des asf-Files bildet die *:hierarchy*-Sektion. Auch sie beginnt mit *begin* und ihr Abschluss wird durch *end* gekennzeichnet.

Zeilenweise werden hier Parent-Child-Beziehungen der Skelettstruktur angegeben. Das bedeutet, es wird festgelegt, mit welchen Abhängigkeiten von einander die Knochen sich später bewegen werden. Den Anfang macht immer der Rootbone. Jede Zeile beginnt mit dem Namen des Parent-Bones gefolgt von einem oder mehreren Child-Bones. Jeder Bone, außer dem Root hat genau einen Parent-Bone, aber ein Elternknochen kann über mehrere Children verfügen.

```
:hierarchy
 begin
  root lhipjoint rhipjoint lowerback
  lhipjoint lfemur
  lfemur ltibia
  ltibia lfoot
  lfoot ltoes
  rhipjoint rfemur
  rfemur rtibia
  rtibia rfoot
  rfoot rtoes
  lowerback upperback
  upperback thorax
  thorax lowerneck lclavicle rclavicle
  lowerneck upperneck
  upperneck head
  lclavicle lhumerus
  lhumerus lradius
  lradius lwrist
  lwrist lhand lthumb
  lhand lfingers
  rclavicle rhumerus
  rhumerus rradius
  rradius rwrist
  rwrist rhand rthumb
  rhand rfingers
 end
```

Abbildung 6: Eine typische :hierarchy-Sektion einer asf-Datei

4.4 Aufbau des amc-Files

Der Header des amc-Files umfasst einer Reihe optionaler Angaben, gefolgt von der *:degrees*-Sektion. Hier sind die Freiheitsgrade (dofs) der einzelnen Knochen einzelbildweise aufgeführt. Jedes Frame hat hierbei dieselbe Struktur:

- ein einzelner Integerwert, der die Framenummer angibt
- die Translations- und Rotationswerte in der Reihenfolge wie sie im zugehörigen asf-File definiert wurde
- die einzelnen Bones, mit ihren Freiheitsgraden, wie sie im asf-File definiert wurden.

Jeder Bone steht in einer einzelnen Zeile und jede dieser Zeilen beginnt mit dem Knochennamen, danach stehen in der Zeile die dof-Werte des Bones gemäß den Angaben des asf-Files. Hat ein Bone keine dofs, so erscheint er auch nicht im amc-File.

```
1
root 192.426 854.523 474.51 14.025 -51.8006 -9.52153
lowerback -4.7254 6.58788 6.19511
upperback -0.770013 8.52742 -2.44441
thorax 3.1083 4.34633 -5.26374
lowerneck -29.3449 0.964308 6.57794
upperneck 23.4667 -2.1592 1.71906
head 12.6447 -0.0874887 -0.815106
rclavicle -1.49089e-014 4.13472e-014
rhumerus -20.9449 32.2599 -91.8385
rradius 50.0758
rwrist -6.97745
rhand -2.16278e-012 2.06418e-012
rfingers 7.12502
rthumb 25.5904 -28.3008
lclavicle -1.49089e-014 4.13472e-014
lhumerus -49.9188 -21.0278 84.6523
lradius -0.525533
lwrist 36.5198
lhand -9.03278e-013 1.89243e-012
lfingers 7.12502
lthumb 25.5904 28.3008
rfemur 18.5625 11.8966 33.0396
rtibia 24.6747
rfoot 19.6609 4.2059
rtoes 1.52667e-013
lfemur -43.8146 5.29533 -18.857
ltibia 46.878
lfoot -13.5563 9.7693
ltoes -1.38752e-013
```

Abbildung 7: Darstellung eines Frames in einer amc-Datei

5 Macromedia Director

5.1 Warum Macromedia Director?

Das Multimedia-Autorenprogramm Director von *Macromedia* ist nach Angaben der Firma Macromedia *„…das führende Tool zur Erstellung interaktiver Daten für das World Wide Web, CD-ROM [und] Präsentationen..."*[1]. Director bietet *„…die Möglichkeiten, verschiedene Medien zu integrieren und deren Wiedergabe vom Benutzer steuern zu lassen – also Multimedia zu machen"*[2].

Director ist in seinen verschiedenen Versionen bereits seit 1985 auf dem Markt, damals noch unter dem Namen *VideoWorks.* Bereits in seiner zweiten *VersionVideoWorks Interactive,* kam eine integrierte Skriptsprache hinzu. Aus ihr entwickelte sich die aktuelle Skriptsprache Lingo. Sie ermöglicht es, sehr flexibel auf unterschiedlichste Anforderungen zu reagieren. Angefangen bei der Realisierung von Schaltflächen mit mehreren Zuständen (aktiv, inaktiv, selektiert…) über die Navigation innerhalb der Anwendung bis hin zur Erstellung der Inhalte der Shockwave 3D-Fenster ist alles über Lingo steuerbar. Lingo ist eine der ganz großen Stärken von Director.

Des Weiteren ist Director durch PlugIns erweiterbar. Zahllose PlugIns für die unterschiedlichsten Verwendungszwecke stehen im Internet zur Verfügung, häufig auch als Freeware.

Markus Eberl und Jens Jacobsen führen in ihrem Buch „Macromedia Director 8.5" folgende Stärken von Director an:

„- *Tempo: Die Geschwindigkeit von Director ist von anderen Autoren-Programmen unübertroffen. […] Programmierer bemerken […] durchaus anerkennend, dass Lingo-Skripte bis zu drei Viertel der Geschwindigkeit von Programmen in der Sprache C erreichen.*

- *Flexibilität: Lingo wurde inzwischen zu einer Skriptsprache ausgebaut, mit der man fast alle Aktionen am Computer steuern kann, Und die Dinge, die nicht möglich sind, können als Xtras realisiert werden. […]*

- *Cross-Plattform-Kompatibilität: Die Entwicklung von Anwendungen für PC und Apple Macintosh ist mit Director sehr einfach. […]*

---

[1] *Gross, Phil und Mike*, Director 8.5, Shockwave Studio für 3D, 2002, S. 1

[2] *Eberl, Markus/Jacobsen, Jens,* Macromedia Director 8.5 – Das komplette Wissen für Multimedia-Publisher, 2002, S. 46

- *Lizenzfreier Vertrieb: Wer das Programm Director gekauft hat, der darf so viel damit erstellte Anwendungen vertreiben, wie er will. […]*
- *Weite Verbreitung: Mittlerweile gibt es Bücher zu Director für jeden Geschmack, in Zeitschriften werden Tipps und Tricks für das Programm veröffentlicht, es gibt Anwender-Stammtische – echte wie virtuelle -, Schulungen werden angeboten und im Internet bietet Macromedia Anwender-Support. […] Laut Macromedia gibt es weltweit über 350 000 Menschen, die mit Director arbeiten.“*[3]

Das Arbeiten in Director gestaltet sich sehr übersichtlich. Alles, was für eine Anwendung benötigt wird, Texte, Grafiken, 3D-Darsteller, Lingo-Skripte etc. werden als Darsteller, beziehungsweise Member, in einer Besetzungsliste aufgeführt.

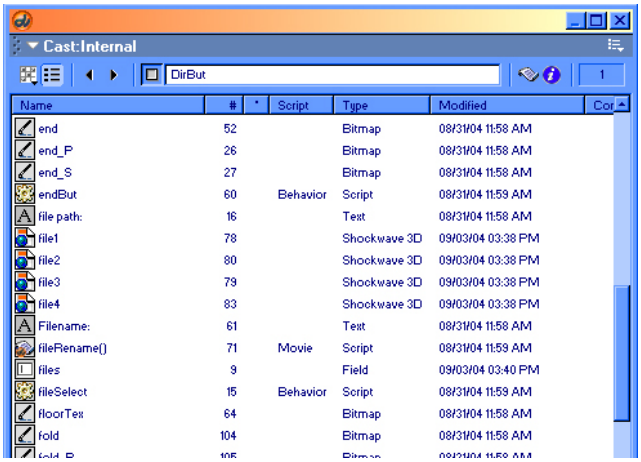

Abbildung 8: Das Cast-Fenster

Alle Darsteller dieser *Cast* sind mit einem kleinen Icon versehen, um ihren Typus schneller identifizieren zu können. Jeder Darsteller kann im eigentlichen Director-„Film“ mehrfach verwendet werden. Die Cast kann nach verschiedenen Vorgaben sortiert werden.
Neben dem Cast-Fenster gibt es noch die Bühne. Sie entspricht dem, was später in der fertigen Anwendung zu sehen ist. Hier werden die Darsteller räumlich platziert.

[3] *Eberl, Markus/Jacobsen, Jens*, Macromedia Director 8.5 – Das komplette Wissen für Multimedia-Publisher, 2002, S. 44f

Ihre zeitliche Anordnung wird im so genannten Score-Fenster festgelegt.

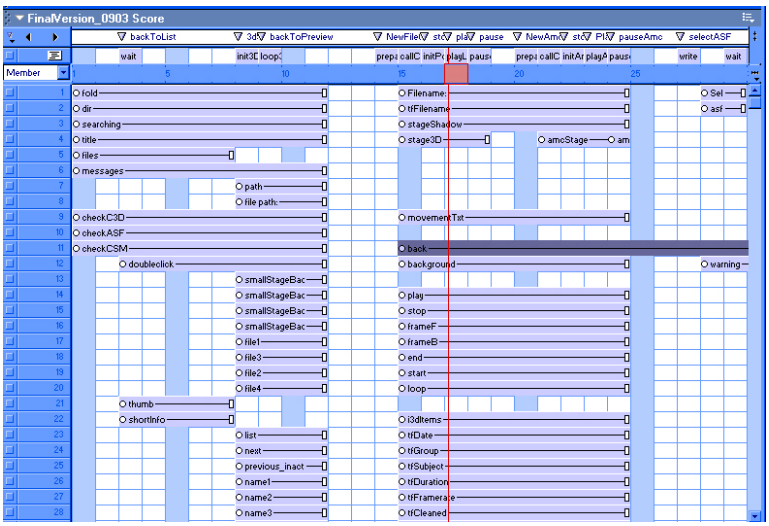

Abbildung 9: Das Score-Fenster

Wird nicht mit Lingo-Skripten in den zeitlichen Ablauf eingegriffen, so wird der Reihe nach der Inhalt der einzelnen Frames, also alle Darsteller, die sich an dieser Stelle im Score-Fenster befinden, in der Bühne dargestellt. Normalerweise wird aber der Ablauf über Skripte gesteuert. Das gezielte Anspielen einer Stelle in der Zeitleiste wird durch das setzen von Markern vereinfacht. Verschiebt sich ein ganzer Abschnitt zeitlich, so muss nur der Marker mit verschoben werden und nicht in sämtlichen Skripten die entsprechende Framenummer geändert werden.

Durch diese drei Fenster ist es möglich, auch bei größeren Projekten, wie dem vorliegenden, stets den Überblick zu bewahren. Dadurch und durch eine Reihe weiterer Hilfsmittel verfügt Director nicht nur über viele Fähigkeiten, sondern ermöglicht auch noch ein angenehmes Arbeiten.

5.2 Shockwave 3D

Seit der Version 8.5 verfügt Director über eine Erweiterung zur 3D-Fähigkeit. Jetzt ist erstmals die Darstellung skalierbarer Echtzeit-3D-Grafik möglich. Mit Hilfe entsprechender PlugIns können 3D-Inhalte wie gewohnt mit 3D-Software-Paketen wie Aliaswavefront Maya oder discreet 3DStudio Max erstellet werden und in das von Director verwertbare w3d-Format umgewandelt werden. Anschließend können sie in Director als Darsteller importiert werden. Diese Szenen können nachträglich in Director modifiziert werden. So ist es beispielsweise ohne weiteres möglich, zusätzliche Kameras und Lichter einzusetzen oder bestehende Lichter zu verändern.

Es ist auch möglich, die Daten in Form von obj-Dateien zu transferieren. Hierfür stellt Macromedia auf ihrer Internetseite ein entsprechendes PlugIn zur Verfügung.

Darüber hinaus stellt Shockwave 3D auch einige Basisobjekte wie Kugeln, Quader und Ebenen zur Verfügung, die nach eigenen Vorstellungen erzeugt, texturiert und animiert werden können. Auch für die Erstellung und Steuerung von Lichtern und Kameras steht eine breite Auswahl von Lingo-Kommandos zur Verfügung.

Eine großartige Hilfe für die Arbeit mit Shockwave 3D bietet der 3D Property Inspector (3DPI) von ullula, alias Ursula Gusenbauer. Sie entwickelte ihn in Absprache mit den Entwicklern und Nutzern der Betaversion von Director 8.5, da der bereits bestehende Eigenschaftsinspektor, den Director bereithält, für die Darstellung der riesigen Informationsmenge einer 3D-Szene nur bedingt geeignet ist. Der 3DPI ist auch als Shareware verfügbar und kann im Internet[4] kostenlos heruntergeladen werden.

5.3 Xtras - Director PlugIns

Eine weitere Stärke von Director istdie einfache Erweiterbarkeit der Funktio-nalität des Programms mit eigenen Zusatzprogrammen. Diese PlugIns werden in Director als Xtras bezeichnet. Seit der Version 5.0 sind sie fester Bestandteil von Director. Sie haben sich aus den wesentlich ein-geschränkteren XObjects entwickelt. Während XObjects zwar beispielsweise Befehle an das Betriebs-system übermitteln und Informationen abrufen konnten, waren sie nicht dazu in der Lage, eine Kontrollfunktion innerhalb des Directorfilms zu übernehmen. Sie glichen einer Einbahnstraße. Seit der Einführung der Macromedia Open Arcitecture (MOA) hat sich dies geändert und der „Verkehr“ ist nun in beiden Richtungen möglich.

Mit der MOA ist es möglich, programmübergreifende PlugIns zu schreiben. PlugIns für Director können so auch für andere Macromedia Produkte, wie beispielsweise Authorware verwendet werden.

Standardmäßig sind bereits über 60 Xtras wie zum Beispiel das FileIO bei der Installation von Director bereitgestellt. Andere, wie beispielsweise das für diese Arbeit verwendete FileXtra4, können im Internet kostenlos heruntergeladen werden.

Markus Eberl und Jens Jacobsen fassen mehrere unterschiedliche Technologien unter dem Begriff „Xtra“ zusammen, die überwiegend nichts mit der MOA zu tun haben. Wird „PlugIn“ definiert als ein Programm, das mit anderen Anwendungen interagiert, um eine spezielle Funktion bereit zu stellen, so lassen sich als echte PlugIns nur MOA-Xtras bezeichnen, daher bezieht sich im weiteren Text der Begriff Xtra ausschließlich auf MOA-Xtras. Aber auch bei diesen „echten“ Xtras lassen sich noch vier verschiedene Arten unterscheiden:

[4] http://www.ullala.at/3DPI, Stand 2004

> *MOA ist eine Software-Schnittstelle, die es Entwicklern erlaubt, Director durch Zusatzmodule zu erweitern. Diese Zusatzmodule werden üblicherweise in der Programmiersprache C oder C++ geschrieben. ... Durch ihre Funktion unterscheidet man zwischen Script-, Darsteller-, Übergangs- und Werkzeug-Xtras.*[5]

Der für diese Arbeit entscheidende Typus ist das Script-, oder auch Lingo-Xtra. Mit ihm ist es möglich, eigene Lingo-Kommandos zu entwickeln.

Wenn in Director im Script- oder im Message-Fenster der fünfte Lingobutton in der Befehlszeile (*Scripting Xtras*) gedrückt wird, wird eine Liste aller verfügbaren Script-Xtras angezeigt und im jeweiligen Untermenü werden die dazugehörenden Kommandos angezeigt.

5.3.1 Erstellen eines Script Xtras am Beispiel das amcReader-Xtras

Zur Erstellung von Script-Xtras wird das Xtra Development Kit (XDK) benötigt. Es steht auf der Macromedia-Seite zum Download zur Verfügung[6]. Neben den benötigten Include-Dateien, der Dokumentation der bereitgestellten Klassen und Beispiele, enthält es auch Grundgerüste für die verschiedenen Xtras, die vom Programmierer nur noch angepasst werden müssen.

Theoretisch klingt das alles sehr einfach, aber bei näherer Betrachtung sind diese Grundgerüste und die Dokumentation leider ein wenig verwirrend. Tab Julius war diese Problematik anscheinend bekannt. Darum veröffentlichte er Anfang des Jahres in mehreren Ausgaben der Zeitschrift *MX developer's journal* Artikel zum Thema Director Xtras und ihre Erstellung. Diese Artikel sind ein guter Einstieg in die Welt der MOA.

Für das Skript-Xtra stehen zwei verschiedene Grundgerüste zur Verfügung, die beide genutzt werden können. Da aber für diese Diplomarbeit ausschließlich das Grundgerüst aus dem Skeleton-Ordner verwendet wurde, beziehen sich alle weiteren Ausführungen nur auf dieses Beispiel. Auch Julius bezieht sich in seinen Ausführungen darauf.

Wenn der Arbeitsbereich im Microsoft Visual Studio geöffnet wird ist zu erkennen, dass das Projekt bereits aus mehreren Dateien besteht. Von Interesse für die Entwicklung eigener Xtras sind vier davon: Cregster.h, Cregster.cpp, Cscript.h und Cscript.cpp. Nach Julius' Angaben sollte beim Erstellen eigener Projekte auch noch die Datei Xtra.cpp mitkopiert werden.

---

[5] *Eberl, Markus, Jacobsen, Jens,* Macromedia Director 8.5 – Das komplette Wissen für Multimediapublisher, 2002, S.542

[6] http://www.macromedia.com/Support/xtras/xdks/dmx_xdk.html, Stand 2004

Für das weitere Vorgehen ist es hilfreich, zunächst einmal alle irreführenden Kommentare zu löschen.

Im nächsten Schritt wird der Dateipfad des Include-Ordners des XDK unter *Extras/Optionen* bei den Include-Verzeichnissen hinzugefügt. In den Projekteinstellungen kann der Ordner, der die Director-Xtras enthält, als Ausgabepfad für die kompilierte Datei angegeben werden. Das erspart eine Menge Kopierarbeit.

Nachdem diese Vorarbeiten geleistet sind, kann die eigentliche Programmierarbeit beginnen. Zunächst einmal werden die bereits vordefinierten Klassen als eigene Dateien dem Projekt hinzugefügt. In diesem Fall sind das acclaim.h, acclaim.cpp, asfReader.cpp und asfReader.h. Die Header-Dateien werden mit einem #include Befehl in das Projekt eingebunden:

```
#ifndef _H_acclaim
    #include "acclaim.h"
#endif
```

Um die Xtras auseinander halten zu können, benötigt Director in den Header-Dateien des Xtras eigene Globally Unique Identifier Nummern (GUID). Jedes Xtra benötigt seine eigenen GUIDs, andernfalls wird Director die Fehlermeldung bringen, dass doppelte Xtras vorliegen. Die GUID ist eine 128-bit-Zahl, die beispielsweise mit dem Programm Guidgen.exe, das im Microsoft Visual Studio (Tools-Ordner) enthalten ist, erzeugt werden kann. Sowohl für die CRegster.h- als auch für die CScript.h-Datei wird jeweils eine eigene GUID benötigt. Dafür wird folgende Zeile ersetzt:

```
#error PLEASE DEFINE A NEW CLSID
```

Der neue Eintrag sieht dann in etwa wie folgt aus:

```
// {90D62442-5D22-4bd4-A3D9-C1F54EF9BE3F}
DEFINE_GUID(CLSID(CRegister),
0x90d62442, 0x5d22, 0x4bd4, 0xa3, 0xd9, 0xc1, 0xf5, 0x4e, 0xf9, 0xbe, 0x3f);
```

In der Datei CScript.h wird der Eintrag „CRegister“ durch „CScript“ ersetzt.

Die nächste wichtige Änderung erfolgt in CRegster.cpp. Hier muss die so genannte „Message Table“ angepasst werden.

```
/* MODIFY: */

static char msgTable[] = {
    "xtra ScriptingSkeleton\n" \
    "new object me\n" /* standard first handler entry in all message tables */
    "-- Template handlers --\n"
    "* globalHandler -- prints global handler message\n"
    "+ parentHandler object xtraRef -- prints parent handler message\n"
    "childHandler object me -- prints child handler message\n"
    };
```

Die modifizierte Message Table sieht in diesem Fall so aus:

```
static char msgTable[] = {
    "xtra amcReader\n" \
    "--by Cosima Schunk\n\n"
    "new object me\n"
    "amcOpen object me, string filenameAMC, string filenameASF \t-- opens and reads
                                                                    an amc file\n"
    "asfAngle object me \t\t\t-- returns integer: 1 - deg, 2 -rad \n"
    "asfHierarchy object me \t\t-- returns list of indices of bones in hierarchy\n"
    "asfLength object me, int index \t-- returns list of lengths of bones\n"
    "amcFps object me \t\t-- returns frames per second\n"
    "amcDuration object me \t\t\t-- returns number of frames\n"
    "amcNBones object me \t\t-- returns number of bones\n"
    "amcDirections object me, int index \t-- retruns list with start orientations\n"
    "amcAxis object me, int index \t\t-- returns list with axis orientations\n"
    "amcData object me \t\t\t-- retruns list of rotation data of all bones\n"
    "asfName object me, int index\t\t-- returns bone's name\n"
    "asfUnit object me, \t\t\t-- returns unit's length\n"
    "nameCompare object me, \t\t -- returns 1 if files are appropriate\n"
    };
```

Die Einträge der Message Table sind in Director über das Message Window abrufbar, wenn in Director der *put interface*-Befehl eines Xtras ausgeführt wird.

Der Eintrag „ScriptingSceleton“ in der zweiten Zeile des Message Tables wird durch den Namen des Xtras ersetzt.

Die dritte Zeile „new object me\n“ ist immer vorhanden. Danach werden die übrigen Kommandos des Xtras aufgeführt. Kommandos die mit einem Asterisk ’*’ beginnen, sind globale Kommandos. Sie können in Director ausgeführt werden, ohne dass mit dem Befehl *new* eine Instanz des Xtras erzeugt werden muss.

Jedes Kommando beginnt mit einem Kommandonamen gefolgt von möglichen Argumenten, die vom rufenden Programm übergeben werden. Als Argumente kommen integer, float, string, symbol, object, any oder ’*’ in Frage. Der Asterisk sollte mit Vorsicht verwendet werden. Er ermöglicht die Übergabe einer beliebigen Anzahl von Argumenten belieben Typs. Child-Funktionen müssen zwingend über eine Instanz des Xtras aufgerufen werden und können daher auch als erstes Argument nur ein Object übergeben bekommen.

Jedes Argument kann außerdem mit einem Kommentar versehen werden. So hat es der Anwender einfacher, zu wissen welche Eingabe von ihm erwartet wird.

```
"amcDirections object me, int index \t-- retruns list with start orientations\n"
```

Jeder Kommandoname sollte zu Beginn einen möglichst sinnvollen Namenszusatz haben, um eindeutig zugeordnet werden zu können. Wenn zwei Xtras über das Kommando *open( )* verfügen, ist Director nicht in der Lage zu erkennen, welches Xtra gemeint ist. Wird aber in einem Xtra *amcOpen( )* und in dem anderen *csmOpen( )* verwendet, so ist der Aufruf eindeutig.

Nun werden zunächst in der Datei CScript.h unter der Methode *EXTERN_DEFINE_METHOD(MoaError, Call, (PMoaMmCallInfo))* die Funktionen definiert, die den aus Director aufgerufenen Kommandos entsprechen:

```
EXTERN_BEGIN_DEFINE_CLASS_INTERFACE(CScript, IMoaMmXScript)
    EXTERN_DEFINE_METHOD(MoaError, Call, (PMoaMmCallInfo))

  private:
    EXTERN_DEFINE_METHOD(MoaError, amcOpen,          (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, asfAngle,     (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, asfHierarchy, (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, asfLength,        (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, amcFps,           (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, amcDuration,      (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, amcNBones,        (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, amcDirections,    (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, amcAxis,          (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, amcData,          (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, asfName,          (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, asfUnit,          (PMoaDrCallInfo callPtr))
    EXTERN_DEFINE_METHOD(MoaError, compareNames, (PmoaDrCallInf CallPtr))
EXTERN_END_DEFINE_CLASS_INTERFACE
```

Etwas weiter unten in dieser Datei befindet sich eine enum-Tabelle. Wird das Xtra aus Director heraus gestartet, geschieht dies über die *::Call*-Funktion. Ihr wird eine Zahl übergeben, die angibt, welches Kommando der Message Tabel aufgerufen wurde. Das erste Kommando „new“ ist immer #0. Durch dieses Zahlensystem wäre es sehr einfach, Fehler zu erzeugen, wenn sich im Programm die Reihenfolge der Kommandos ändern würde. Um den Aufwand zu minimieren, wurde die enum-Tabelle eingerichtet. Wichtig ist dabei, darauf zu achten, dass die Reihenfolge der Einträge der Message Table denen der enum-Tabelle entsprechen.

```
enum
{
m_new = 0,                    /* standard */

    m_amcOpen,
    m_asfAngle,
    m_asfHierarchy,
    m_asfLength,
    m_amcFps,
    m_amcDuration,
    m_amcNBones,
    m_amcDirections,
    m_amcAxis,
    m_amcData,
    m_asfName,
    m_asfUnit,
    m_nameCompare,

    m_XXXX
};
```

Der Eintrag „m_XXXX“ am Ende der Tabelle hat den Zweck, dass jede vom Programmierer eingetragene Zeile mit einem Komma enden kann. So können alle Zeilen beliebig untereinander ausgetauscht werden.

Alles in der MOA ist objektorientiert und somit in Klassen organisiert. Zum Aufruf einer bestimmten Funktion wird ein Zeiger auf das Interface der Klasse, in der sie existiert erzeugt. Über ihn wird die Funktion aufgerufen. Jeder Zugriff auf eine Methode dieser Klasse erfolgt über den Zeiger. Für das Zusammenspiel von C- und Directorvariablen werden zwei Klassen dringend benötigt: *IMoaMmValue* und *IMoaMmList*.

Da Lingo eigentlich keine Typen kennt, existieren alle übergebenen Argumente nur als Values. Auch als Rückgabewert wird nicht wie in C++ ein Float, Integer oder Character sondern ein Value erwartet. IMoaMmValue enthält Methoden, die die „Übersetzung“ ermöglichen: *ValueToFloat( )* wandelt einen Value in ein Float um*, ValueToPoint( )* wandelt einen Value in einen der Größe drei*, IntegerToValue( )* wandelt einen Intergerwert in einen Value…

Mit IMoaMmList können Lingolisten erstellt und bearbeitet werden: *NewListValue( )* legt einen neuen Listeneintrag an*, AppendValueToList( )* fügt einen neuen Wert am Ende einer Liste an *,NewPropListValue( )*legt einen neuen Eintrag in einer Eigenschaftsliste an…

Die Zeiger auf diese Klassen werden ebenfalls in CScript.h erzeugt:

```
/*************************************************************************
 * CLASS INSTANCE VARIABLES
 *************************************************************************/
EXTERN_BEGIN_DEFINE_CLASS_INSTANCE_VARS(CScript)
    PIMoaMmValue        pMmValue;
    PIMoaMmList         pMmList;
    Moa3dCoord          pVector;
EXTERN_END_DEFINE_CLASS_INSTANCE_VARS
```

Auch für die Implementierung der oben festgelegten Funktionen ist in der Datei CScript.cpp das Grundgerüst bereits vorgegeben:

```
/* ------------------------------------------------------ XScrpChildHandler */
MoaError CScript_IMoaMmXScript::XScrpChildHandler(PMoaDrCallInfo callPtr)
{
    UNUSED(callPtr);

    /* variable declarations */
    MoaError err = kMoaErr_NoErr;

    /*
     * --> insert additional code -->
     */

    return(err);
}
```

Die Argumente, die beim Aufruf des Kommandos übergeben werden, sind mit 1 beginnend durchnummeriert. Bei einem Child-Kommando ist das erste Argument ein object. Alle Argumente können über ihre Nummer ausgelesen werden. Dafür steht die Methode *AccessArgByIndex* zur Verfügung:

```
MoaMmValue value;
AccessArgByIndex(ARG_BASE + 1, &value);
```

Im Beispiel des Acclaim-Xtras werden dem Kommando *amcOpen( )* zwei Dateipfade übergeben: die amc- und die asf-Datei. Um von der Open-Methode der Acclaim-Klasse verarbeitet werden zu können, müssen sie in Zeichenketten umgewandelt werden:

```
acclaim *data;

/*------------amcOpen-----------------------*/
MoaError CScript_IMoaMmXScript::amcOpen(PMoaDrCallInfo callPtr)
{
    MoaError err = kMoaErr_NoErr;
    MoaMmValue value;
    MoaChar AMCfile[256] = {0};
    MoaChar ASFfile[256] = {0};
```

```
    data = new acclaim;

    AccessArgByIndex(ARG_BASE + 1, &value);
        pObj->pMmValue->ValueToString(&value,&AMCfile[0],sizeof(AMCfile));
    AccessArgByIndex(ARG_BASE + 2, &value);
        pObj->pMmValue->ValueToString(&value,&ASFfile[0],sizeof(ASFfile));

    data->openAMC(AMCfile, ASFfile);

    return(err);
}
```

Wird ein Wert an Director zurückgegeben, erfolgt dies nicht über den return-Befehl. Dieser dient dazu, innerhalb des Xtras zu prüfen, ob ein Fehler aufgetreten ist. Die eigentliche Wertübergabe an Director erfolgt in der Zeile *callPtr->resultValue*, wobei auch hier alle C-Typen erst in Lingo-values umgewandelt werden müssen:

```
/*------------amcDuration-----------------------*/
MoaError CScript_IMoaMmXScript::amcDuration(PMoaDrCallInfo callPtr)
{
    UNUSED(callPtr);

    MoaMmValue dur;

    MoaError err = kMoaErr_NoErr;

    pObj->pMmValue->IntegerToValue(data->nFrames, &dur);

    callPtr->resultValue = dur;

    return(err);
}
```

Sobald das Xtra kompiliert wurde und im Ordner Macromedia/Director/Xtras liegt, steht es beim nächsten Start von Director zur Verfügung.

5.3.2 Das c3dXtra

Es ist nicht so ohne weiteres möglich, Binärdateien mit Hilfe von Lingo einzulesen[7]. Im Internet werden einige Xtras angeboten, die mit Binärdateien umgehen können. Diese Lösungen sind alle entweder teuer, oder aber zu unflexibel da auch mit ihnen sehr große Teile des Auslesens mit Lingo gesteuert werden, das in diesem Fall wesentlich langsamer arbeitet, als ein C-Programm.

Daher war es nahe liegend, für diesen speziellen Fall ein eigenes Xtra zu schreiben, das von vorne herein die benötigten Funktionen bereitstellt, die zur Darstellung der gespeicherten Markerdaten in Director nötig sind.

Für die spätere Nutzung in Director sind außer der Datensektion nur wenige Informationen relevant: Die Anzahl der Frames, die Anzahl der aufgenom-menen Trajektorien, der Name des Schauspielers (SUBJECT:NAME) und die Abtastrate.

In dem von Macromedia zur Verfügung gestellten Grundgerüsst müssen also nur sechs sehr einfache Funktionen (*c3dOpen, c3dGetData, c3dDuration, c3dNTrajectories, c3dSubject, c3dFrameRate*) und eine globale Variable vom Typ

[7] Beziehungsweise es ist über einige Umwege möglich, aber bei großen Dateien alles andere als effizient. (Anmerkung des Autors)

c3dData eingefügt werden, ferner zwei Zeiger auf MoaMmValue und MoaMmList, mit welchen der Zugriff auf die MOA eigenen Variablen geregelt wird.

Die eigentliche Arbeit findet in den Methoden der c3dData-Klasse statt:

Zunächst wird überprüft, ob die Datei mit dem von Director übergebenen Pfadnamen überhaupt geöffnet werden kann und ob es sich bei der Datei wirklich um ein c3d-File handelt (Kennzahl im Header). Wenn dies der Fall ist, werden in der Funktion *ReadHeader( )* die wichtigsten Informationen ausgelesen und gespeichert. Insbesondere der Start der Parameter- und der Datensektion sind wichtig, da sie für das weitere Auslesen der Datei entscheidend sind.

Nach dem Einlesen des Headers wird der Lesezeiger für die Datei auf das erste Bit der Parametersektion gesetzt und die boolsche Variable *inParamSection* wird auf *true* gesetzt. Eine wichtige Information der Parametersektion ist der Prozessortyp, auf dem die Datei geschrieben wurde. Er entscheidet darüber, ob Floatingpoint-Numbers als Little- oder Big-Endian vorliegen, das heißt ob die Fließ-kommazahlen von rechts nach links oder von links nach rechts gelesen werden müssen. Davon abgesehen ist in diesem Fall nur noch der Name des Schauspielers von Bedeutung. Sobald er ausgelesen ist, wird die Variable *inParamSection false* gesetzt und das Lesen der Parametersektion wird abgebrochen. Falls kein Schauspielername gefunden wird, endet das Auslesen der Parametersektion, sobald ein Gruppen- oder Parametername der Länge 0 gefunden wurde, was das Ende dieser Sektion signalisiert.

Nachdem der Lesezeiger auf den Beginn der Datensektion gesetzt wurde, werden die eingelesenen 3D-Markerdaten einfach der Reihe nach in einen Array von Floats abgelegt. Liegen die Daten als Integer vor, werden sie vorher bereits mit dem in der Headersektion angegebenen Skalierungsfaktor multipliziert. Das Sortieren der X-, Y- und Z-Koordinaten auf die einzelnen Marker erfolgt erst in Director.

Zu beachten ist noch, das alle 8-Bit-Worte, die als Zahl interpretiert werden sollen, als unsigned Character eingelesen werden und dass alle Integerwerte nur 16-Bit umfassen, also als Short Integer gelesen werden müssen.

Um mit der c3dData-Klasse flexibel auf alle Anforderungen eingehen zu können, ist es trotz der wenigen für diese Anwendung benötigten Daten möglich, auch die übrigen Informationen mit den gegebenen Methoden komplett auslesen zu lassen. Es ist also möglich, jeden gespeicherten Parameter mit all seinen Dimensionen unter Berücksichtigung des Formats der abgelegten Daten auszulesen.

5.3.3 Das csmXtra

Das csmXtra ist während dieser Arbeit zeitlich nach dem c3dXtra entstanden. Es wäre auch möglich gewesen, die csm-Dateien, die als ASCII-Files vorliegen, mit Lingo auszulesen. Da Lingo eine Skriptsprache ist, die erst zur Laufzeit interpretiert wird, ist bei großen Dateien der Einsatz eines Xtras in jedem Fall sinnvoller.

Die Markerinformationen der c3d- und der csm-Dateien sind sich sehr ähnlich, ihre Umsetzung in Director ist identisch. Ziel des Xtras war es daher, die Daten so auszugeben, dass die weiterverarbeitenden Lingofunktionen, die in der Anwendung bereits für die Umsetzung der c3d-Dateien bestanden haben, nicht abgeändert werden mussten. Es wurde nur jeweils eine Abfrage eingefügt. Sie überprüft, ob ein Kommando des c3d- oder des csm-Xtras für den Erhalt der gewünschten Daten aufgerufen werden muss.

Auch dieses Xtra benötigt nur einige sehr einfache Funktionen. Im Wesentlichen sind es dieselben, wie im C3d-Xtra, nur die Funktion *csmDate* ist zusätzlich hinzugekommen.

Die größte Herausforderung war, das Auslesen der Sektion *$Point* in einer vertretbaren Geschwindigkeit zu realisieren. Die auch für das Acclaim-Xtra verwendeten Funktionen *deleteLeadingSpaces( )* und *extractDouble( )* erwiesen sich hierfür zu langsam. Am Ende erwies sich die Lösung als effizient, bei der die ausgelesenen Zeilen nicht zwischen rufender und auslesender Funktion hin und her gereicht werden. Stattdessen wird der Vektor, der die Floatwerte enthält direkt gefüllt und einfach ein Zeiger innerhalb der aktuellen Zeile wertweise weiterverschoben. Dabei wird der bereits gelesene Beginn der Zeile nicht gelöscht[8]. Da die Werte ohnehin in der Reihenfolge gespeichert sind, in der sie auch für die Skripte in Director vorliegen müssen, wird dies wie folgt realisiert:

```
void csm::getData(char *line)
{
   double f;
   strtol(line, &line,0);        // lies die Framenummer aus
   while(strlen(line) > 1)
   {
      f = strtod(line,&line);    // lies die folgenden Zeichen als Double aus
      if(f==0)                //falls kein float gelesen wurde -> Dropout...
      {
         data.push_back(0.0);    //...setze die folgenden X,Y und Z koordinaten auf 0.0...
         data.push_back(0.0);
         data.push_back(0.0);
         extractWord(line);      //...und lösche an dieser Stelle das Wort aus der Zeile
      }
      else
         data.push_back(f);
   }
   nFrames++;                    // Zähle die Anzahl der Frames um eins nach oben
}
```

[8] wie es bei den oben genannten Funktionen der Fall wäre (Anmerkung des Autors)

5.3.4 Das AcclaimXtra

Die Erstellung des AcclaimXtras war komplizierter. Das Aufbauen der Skelettstruktur und die Umsetzung der Rotationen erfordert eine viel größere Menge an Informationen als die Umsetzung der Markerpositionsdaten.

Neben der Acclaim-Klasse, die alle Auslese- und Sortierarbeiten übernimmt, enthält dieses Xtra noch zwei weitere Klassen: *bone* und *branch*. In der Bone-Klasse werden sämtliche Informationen gespeichert, die einem bestimmten Bone des Skeletts zugeordnet sind. Die Branch-Klasse dient zur Speicherung der Hierarchie des Skelettes in einer Form, die in Director gut umgesetzt werden kann. Jedes Branch-Objekt umfasst einen einzelnen Zweig der Hierarchie. Die einzelnen Zweige sind in der Acclaim-Klasse durch einen Vektor zum vollständigen Skelett zusammengefasst:

```
vector<branch> hierarchy;
```

Zunächst müssen selbstverständlich zwei Dateinamen beim Aufruf des Open-Kommandos in Director übergeben werden. Da das Xtra speziell für 3DCubic geschrieben wurde, wird vorausgesetzt, dass es sich hierbei immer um ein amc- und um ein asf-File handelt und dass sie auch in dieser Reihenfolge an die Open-Funktion übergeben werden.

Neben allgemeinen Rückgabewerten, wie zum Beispiel die Dauer der 3D-Szene oder der Framerate, die auch bei den anderen beiden Dateiformaten ausgelesen werden, gibt es im Acclaimformat zahlreiche Informationen, die den einzelnen Knochen der Skeletthierarchie zugeordnet sind. Dazu zählen neben dem Namen auch die Länge des Bone, seine Ausrichtung im Raum und die Ausrichtung seiner Rotationsachsen bezüglich des Weltkoordinatensystems. Beim Aufruf der entsprechenden Kommandos in Director muss neben der Referenz auf das amcReader-Objekt auch noch die entsprechende Nummer des Knochens mitgegeben werden.[9]

Die aus der asf-Datei ausgelesene Hierarchie wird als einzelne Liste an Director übergegeben. Diese Liste ist so aufgebaut, dass jeweils ein komplettes branch-Objekt der Hierarchie komplett durchlaufen wird, angefangen mit dem ersten Parentbone bis hin zum letzten Child. Hierbei werden nicht die Bonenamen abgespeichert, sondern jedem Bone ist eine Nummer zugewiesen. 0 entspricht dem Root des Skeletts und die anderen Bones sind mit 1 beginnend durchnummeriert. Ihre Nummerierung entspricht der Reihenfolge, in der sie in dem Vector, der die

[9] Zu beachten ist hierbei, dass die Listen in Director die Nummerierung ihrer Einträge mit 1 beginnen und nicht mit 0 wie es in C++ üblich ist. Auch hier wird vorausgesetzt, dass dies directorseitig berücksichtigt wird.

einzelnen Bone-Objekte enthält, abgelegt wurden. Das Ende eines Zweiges wird mit -1 signalisiert.

```
void acclaim::writeHierarchy()
{
   char *word;
   // durchlaufe die ganze Hierarchie
   for(int i=0; i<hierarchy.size(); i++)
   {
      // durchlaufe die einzelnen Äste der Hierarchie
      for(int j=0; j<hierarchy[i].branches.size(); j++)
      {
         // merke dir den Eintrag an dieser Stelle als normalen sting
         word = (char*)hierarchy[i].branches[j].c_str();
         // Vergleiche Eintrag mit der Liste der Knochennamen…
         for(int k=0; k<names.size(); k++)
            …bei Übereinstimmung: merke dir die KnochenID
            if(strncmp(word, names[k].c_str(), names[k].size()-1)==0)
               ids.push_back(k);
      }
      // ist das Ende des Zweiges erreicht, trage -1 ein
      ids.push_back(-1);
   }
}
```

Darüber hinaus ist es sehr wichtig zu überprüfen, ob das amc- und das asf-File wirklich zusammengehören. Dazu werden die Namen der Knochen im asf-File mit denen des amc-Files verglichen. Sind die Namen der amc-Datei in der Liste der Namen des asf-Files enthalten wird der Wert 1 zurückgeliefert. Andernfalls 0, da in diesem Fall davon ausgegangen wird, dass die beiden Dateien nicht zusammen gehören.

Neben diesen drei selbst geschriebenen Xtras findet zusätzlich folgendes Xtras Verwendung:

FileXtra v4.0.2 das von Kent Kersten als Freeware zur Verfügung gestellt wird[10]. Es dient in der Anwendung in erster Linie dem Durchsuchen der Festplatten und Ordner nach den angegebenen Dateiformaten.

Von Macromedia kommen das MUI- und das FileIO-Xtra zum Einsatz.

Das MUI dient zur Kommunikation mit dem Anwender. Sämtliche Windows-Standard-Dialogboxen der Anwendung werden mit Hilfe des MUI-Xtras realisiert.

Das FileIO-Xtra wird benötigt, um die i3d-Dateien zu erzeugen, sie auszulesen und neu zu schreiben, falls sie erstellt oder geändert wurden.

[10] http://www.kblab.net.xtras, Stand 2004

6 3DCubic

6.1 Aufbau der Anwendung

Die Anwendung setzt sich aus drei Hauptabschnitten zusammen:

- Dem Startscreen, der die Optionen bietet, Laufwerke oder Ordner nach bestimmten Dateiformaten durchsuchen zu lassen. Die Suchergebnisse werden in einer Liste angezeigt. Wird ein Eintrag der Liste selektiert, so werden Informationen über den kompletten Dateipfad, das letzte Änderungsdatum und die Dateigröße angezeigt. Durch Doppelklick auf einen Dateinamen, werden die „Erweiterten Funktionen“ aufgerufen. Außerdem hat der Anwender von hier aus die Möglichkeit, zum Preview-Screen zu wechseln.

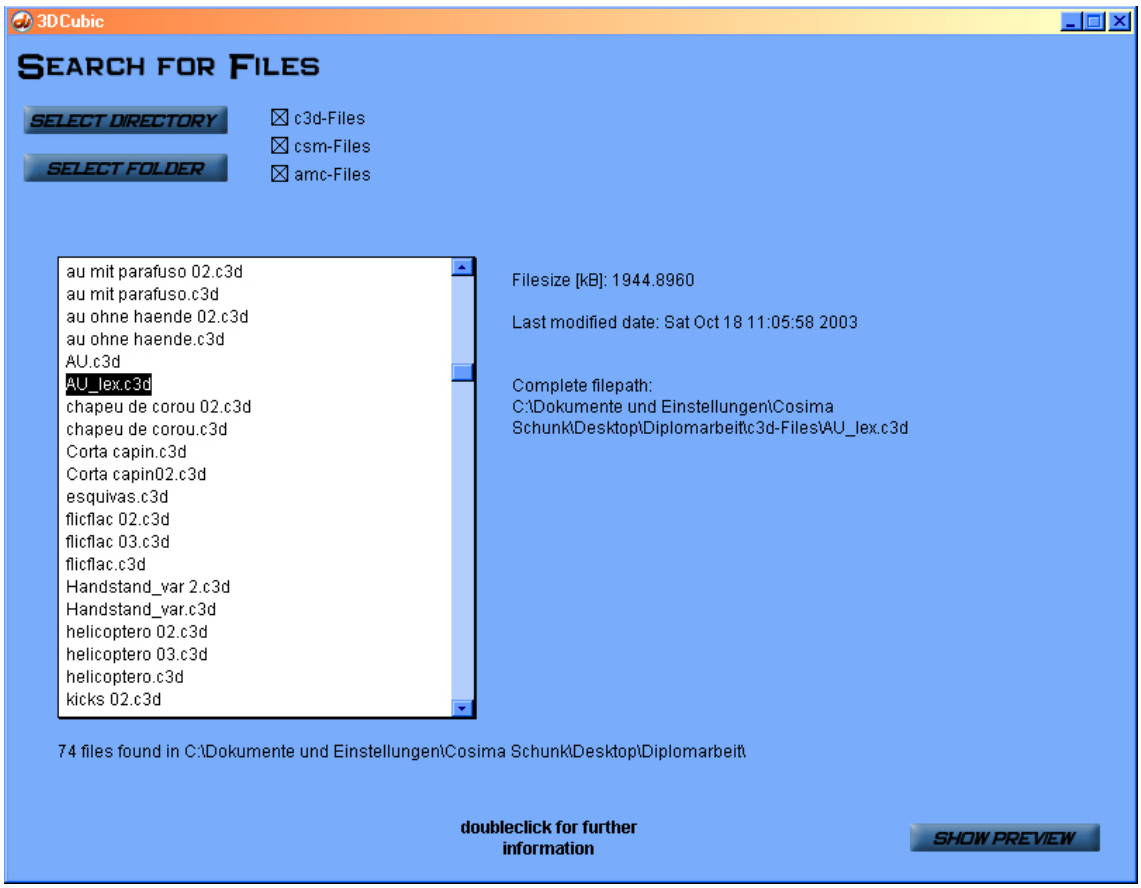

Abbildung 10: Startscreen

- Dem Preview-Screen der auf den Suchergebnissen der vorherigen Seite aufbaut. In ihm sind vier kleine Shockwave3D-Bühnen platziert, in denen jeweils vier Dateien in der 3D-Ansicht dargestellt werden. In der ersten Bühne wird jeweils die Datei angezeigt, die gerade in der Liste des Startscreens markiert war. In den übrigen Bühnen werden die nachfolgenden

Einträge der Liste dargestellt. Der Anwender hat die Möglichkeit, sich nach und nach alle Listeneinträge anzeigen zu lassen. Durch Doppelklick auf eine der 3D-Bühnen kommt er auch von hier aus zu den „Erweiterten Funktionen“ oder er kann zur Listenansicht zurückkehren.
Es besteht die Möglichkeit, einen neuen Suchauftrag zu starten.

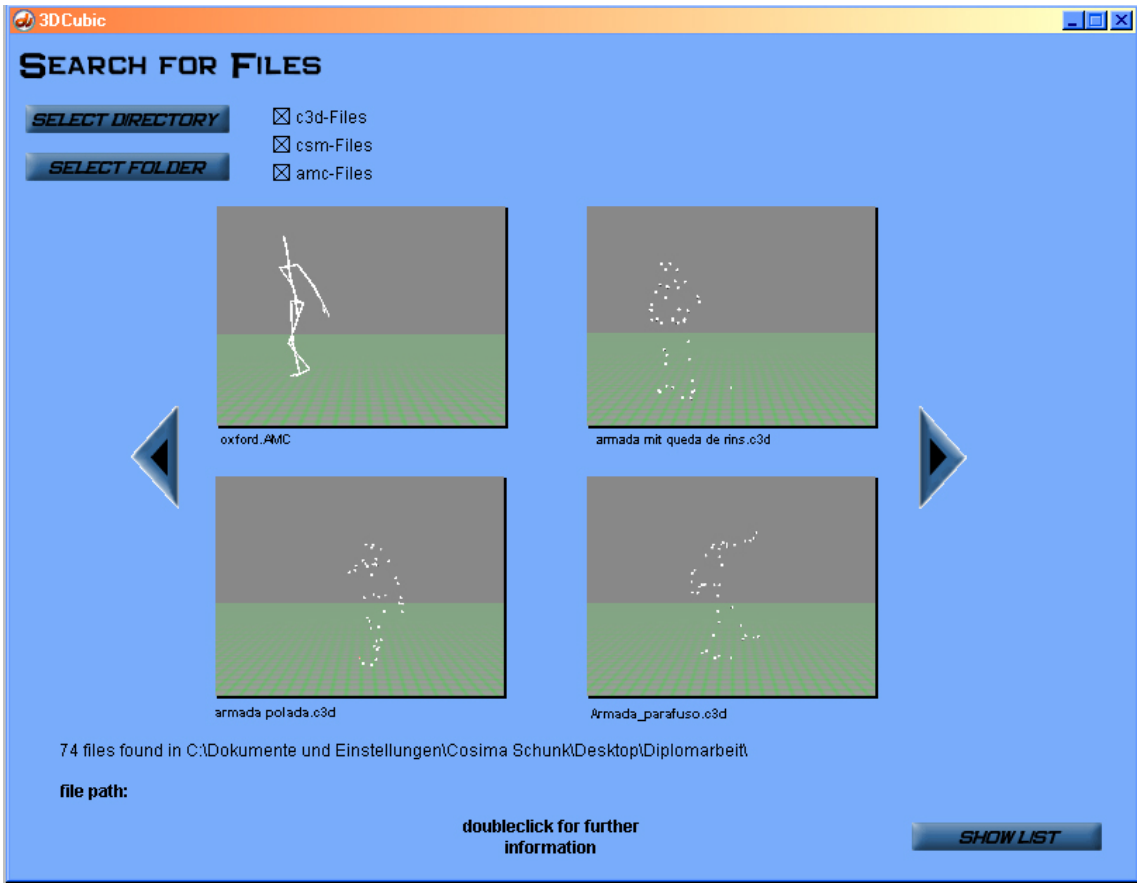

Abbildung 11: Preview-Screen

- Die „Erweiterten Funktionen“ bieten dem Anwender zum einen die Möglichkeit, sich die 3D-Szene genauer zu betrachten. Er hat dabei die üblichen Schaltflächen einer Videosteuerung zur Verfügung und kann innerhalb der 3D-Bühne navigieren, das heißt ein- und auszoomen, die Bühne drehen oder verschieben.

  Außerdem hat er die Möglichkeit, die Inhalte des zugehörigen Informationsfiles einzusehen und sie zu editieren. Falls kein Informationsfile vorliegt, wird der Anwender beim Betreten dieses Fensters darauf hingewiesen und ihm wird angeboten, es zu erstellen. Über die Schaltfläche *Back* kommt er zurück zum vorherigen Screen.

Falls der Anwender diesen Screen durch die Auswahl einer amc-Datei aufruft, kommt er zuvor noch zu einem Zwischenscreen. Hier wird er aufgefordert, ein zugehöriges amc-File aus der Liste auszuwählen.

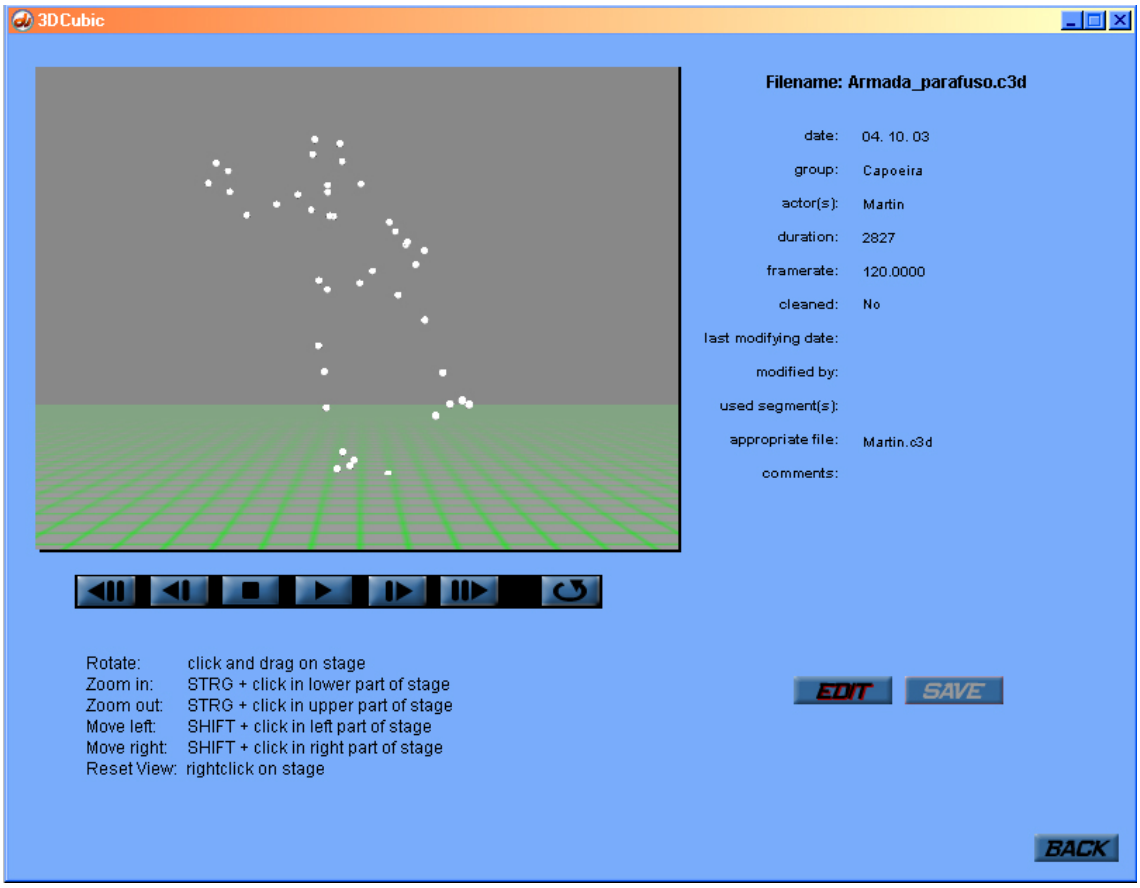

Abbildung 12: Erweiterte Funktionen

6.2 Umsetzung der Marker-Daten

Zur Erstellung der 3D-Ansichten muss zunächst ein c3dReader- beziehungsweise ein csmReader-Objekt erzeugt werden, das es ermöglicht, mit Hilfe der entsprechenden Xtras auf die Inhalte der Dateien zu zugreifen.

Der erste Schritt nach dem Öffnen und Auslesen der Datei durch das Xtra ist das Zurücksetzen der 3D-Welt. Dies ist notwendig, damit wirklich alle evtl. zuvor erzeugten Inhalte gelöscht werden. Gleichzeitig wird gewährleistet, dass keine doppelten Objektnamen auftreten, da dies bei dem Versuch eines Zugriffs zum sofortigen Absturz der Anwendung führen würde. Das Zurücksetzen der 3D-Welt muss auf jeden Fall erfolgen, bevor der 3D-Darsteller tatsächlich abgespielt wird. Andernfalls gehen einigen Funktionen, insbesondere der Kamerasteuerung im Erweiterte-Funktionen-Screen, Informationen verloren. Dies hätte ebenfalls einen Programmabsturz zur Folge.

Beide Marker-Formate werden im Folgenden gleich behandelt, da die Rückgabewerte der Xtras dasselbe Format haben. Trotzdem muss immer überprüft werden, ob ein c3d- oder ein csm-File benutzt wird. Nur so wird sichergestellt, dass das richtige Kommando verwendet wird, um beispielsweise Informationen über Dauer oder Abspielgeschwindigkeit zu erhalten.

Zunächst erfolgt das Verteilen der langen Liste mit den Positionsinformationen auf einzelne Listen für die einzelnen Marker. Es ist in Director möglich, so genannte Property-Lists, strukturierte Listen mit Eigenschaftsnamen, anzulegen. Genau genommen sind Property-Lists eine Zusammenfassung von untergeordneten Listen zu einer übergeordneten. Auf die untergeordneten Listen wird über den Eigenschaftsnamen zugegriffen. Da Lingo wesentlich langsamer arbeitet, als C++, nimmt dieser Vorgang leider einige Zeit in Anspruch. Bei einer neuen Version der Anwendung sollte daher in Erwägung gezogen werden, ob es nicht möglich ist, die Positionsdaten bereits vom Xtra markerweise sortieren zu lassen.

```
on createList (whichMember, obj, fType)
 -- reset 3D-World
 member(whichMember).resetWorld()

 …

 Marker1 =[#XP:[] ,#YP:[] ,#ZP:[]]
 MarkersList.append(Marker1)

 -- create own list for every marker
 repeat with k = (nTraj-1) down to 1
  append(MarkersList, Marker1.duplicate())
  n = n+1
 end repeat

 -- fill lists
 repeat with i in MarkersList
  zet = zaehler
  repeat with j=1 to dur-1
   append i.XP, (d.getAt(zet))
   zet = zet+(nTraj*3)
  end repeat

  zet = zaehler+1
  repeat with j=1 to dur-1
   append i.YP, (d.getAt(zet))
   zet = zet+(nTraj*3)
  end repeat

  zet = zaehler+2
  repeat with j=1 to dur-1
   append i.ZP, (d.getAt(zet))
   zet = zet+(nTraj*3)
  end repeat

  zaehler = zaehler+3
 end repeat
 …

end
```

Nun werden die einzelnen Marker erzeugt. Die Markerobjekte werden ebenfalls in Listen zusammengefasst und später mit ihrem Index in der Liste angesprochen.

Um in Shockwave 3D ein Standardobjekt zu erzeugen, muss zunächst eine *Ressource* für dieses Objekt erzeugt werden, in welcher die Eigenschaften des Objektes festgelegt werden.

```
MRes = member(whichMember).newModelResource("RMarker",#sphere)
MRes.radius = 20
```

Um das Objekt selbst zu erzeugen muss ihm eine Ressource und ein in dieser 3D-Welt einmaliger Name zugewiesen werden.

```
member(whichMember).newModel("Marker", MRes)
```

Auch der Shader und die Textur werden erst unabhängig von dem Objekt erzeugt, bevor sie dem Objekt zugewiesen werden.

```
mShdr = member(whichMember).newShader("s2",#standard)
mTx = member(whichMember).newTexture("tx2",#fromCastMember, mem-
                                                    ber("markerTex"))
mShdr.textureList[1] = mTx
mShdr.specular = rgb( 0, 0, 0 )
member(whichMember).model("Marker").shaderList = mShdr
```

Zusätzlich werden noch eine Ebene als Boden sowie eine Kugel als Markierung des Ursprungs des Weltkoordinatensystems erzeugt. Für eine bessere Ausleuchtung wird das standardmäßig bereitgestellte Licht durch zwei eigene Lichtquellen ersetzt. Als Kamera wird die Default-Kamera verwendet, die automatisch mit jedem 3D-Darsteller erzeugt wird.

Um dasselbe Kommando für die kleinen Bühnen des Previewfensters und für die große Bühne der erweiterten Funktionen verwenden zu können, werden beim Kommandoaufruf der Name des Members, das entsprechende Xtra-Objekt und der Datentyp der darzustellenden Datei angegeben. Der Rück-gabewert ist eine global zugängliche Liste der Marker.

```
Points = createPoints("stage3D", c3dData, "c3d")
```

Die Schleife für das Abspielen der Animation ist in diesem Fall sehr einfach. Bei den kleinen Fenstern des Preview-Screens wird die Abspielgeschwindigkeit auf den festen Wert von 10 Bildern pro Sekunde und bei der großen Bühne auf 30 Bildern pro Sekunde gesetzt. Es wird immer dasselbe Frame des Directorfilmes durchlaufen. Bei jedem Durchlauf wird die globale Variable für die aktuelle Zeit im 3D-Film um den Wert *factor = fps der Datei/fixe fps des Directorfilms* erhöht. Ist der aktuelle Wert der globalen Variablen größer als die Gesamtframezahl der Datei, wird er wieder auf 1 gesetzt. Dann werden die einzelnen Markerpunkte auf die Position gesetzt, die an der Stelle der globalen Variablen in der entsprechenden Positionsliste angegeben ist.

```
if (fNum < dur-1) then
 repeat with i in Markers
  newPos = vector((i.XP[fNum]),(i.YP[fNum]),(i.ZP[fNum]))
  if (i.XP[fNum]=0)then
   Points[counter].visibility = #none
  else
   Points[counter].visibility = #front
  end if
  Points[counter].transform.position = newPos
  counter = counter +1
 end repeat
end if
```

Hat ein Markerpunkt die X-Koordinate von exakt 0.0, so wird davon ausgegangen, dass er beim Capturen nicht von genügend Kameras erfasst wurde und die Positi-

on somit in der Datei auf einen Default-Wert oder überhaupt nicht gesetzt war. Seine Sichtbarkeit wird auf *#none* gesetzt.

6.3 Umsetzung der Rotations-Daten

Auch in Director ist die Arbeit mit den Rotationsdaten komplizierter als die Umsetzung der Positionsdaten.

Die erste Herausforderung ist, dass immer zwei Dateien vorhanden sein müssen, um die hierarchische Struktur und die Rotationen der Knochen darstellen zu können. Vor der Darstellung in den Erweiterten Funktionen ist dies sehr einfach: die amc-Dateien werden genau wie die c3d- oder die csm-Dateien in der Auswahlliste aufgeführt. Werden dann die Erweiterten Funktionen aufgerufen, so erscheint eine weitere Liste, in der die asf-Dateien zur Auswahl stehen. Ist die Liste leer, so wird der Anwender darauf hingewiesen. Er kann eine neue Suche starten, die Ordner mit einschließt, in der auch asf-Dateien zu finden sind.

Auch für die Darstellung im Preview-Fenster wird ein asf-File benötigt. Da es sich um eine automatisch erstellte Vorschau handelt, ist es nicht möglich, den Anwender für jede einzelne amc-Datei nach der zugehörigen asf-Datei zu fragen. Obwohl es in der Theorie möglich sein sollte, aus dem amc-File den Namen des passenden asf-Files auszulesen, werden in der Praxis die Dateien später meistens umbenannt, ohne dass die entsprechende Zeile in der amc-Datei geändert wurde. Falls der Name der zugehörigen asf-Datei im entsprechenden i3d-File gespeichert wurde, kann diese Datei verwendet werden, ansonsten wird auf den ersten Eintrag der asf-Liste zurückgegriffen.
Doch auch wenn diese Liste leer ist, sollte eine Darstellung im Preview-Fenster möglich sein. In diesem Fall wird dem amcReader-Xtra für den zweiten Dateinamen, das asf-File, ein leerer String übergeben. Das Xtra füllt die Variablen, die normalerweise aus dem asf-File ausgelesen würden, mit Standardwerten. In ihrer Form entsprechen diese einer asf-Datei, wie sie von Vicons BodyBuilder erstellt wird.

Leider kann es sowohl mit dem ersten Listeneintrag als auch mit Defaultwerten zu falschen Darstellungen kommen, da der Skalierungsfaktor, der im asf-File in der *:units*-Sektion gespeichert ist, nicht zwingend dem Faktor der Originaldatei entspricht. Als Folge werden zwar das Skelett und die Rotationen korrekt wiedergegeben, aber die Postitionsinformationen des Root-Bone werden mit dem falschen Faktor multipliziert. Ist der Faktor zu klein, bewegt sich das Skelett scheinbar auf der Stelle. Oder aber, der Faktor ist so groß, dass sich die Position des Skeletts außerhalb des Bildausschnittes befindet.

Das nächste Problem, das sich bei der Darstellung einer Skeletthierarchie ergibt, ist die Tatsache, dass es in Director nicht möglich ist, echte Joint- oder Bone-

Objekte zu erzeugen, wie sie aus Computeranimationssoftware wie beispielsweise Aliaswavefront Maya oder Discreet 3DStudio Max üblich sind. Ein Joint in Maya ist eigentlich eine kleine Kugel, die positioniert und rotiert werden kann. Das Verbindungsstück zum übergeordneten Joint ergibt sich dabei automatisch. Wird der übergeordnete Joint bewegt, bewegt sich der untergeordnete Rest der Hierarchie entsprechend mit.

6.3.1 Ansatz 1: Darstellung der Skelettstruktur durch Quader

Die Idee hinter diesem Ansatz ist, die Bones, beziehungsweise die Verbindung der Bones untereinander durch Quader darzustellen, die in Director selbst erzeugt werden.

Die erste Schwierigkeit stellt sich beim Zugriff auf den Pivotpunkt des Quaders. Der Pivotpunkt ist der Punkt, um den die Rotation des Objekts erfolgt. Auch die Positionskoordinaten der Objekte beziehen sich auf ihn. Bei der Erzeugung in Director liegt er in der Mitte des Quaders.
Für die Erstellung der Hierarchie ist es sinnvoller, wenn er am oberen, beziehungsweise unteren Ende liegen würde. Es ist nicht möglich, mit Lingo diesen Pivotpunkt unabhängig vom eigentlichen Objekt zu manipulieren. Glücklicherweise bietet Lingo die Möglichkeit auf die einzelnen Meshes, aus denen ein 3D-Objekt besteht und deren einzelne Vertices zuzugreifen. Um den Pivotpunkt vom Zentrum des Quaders auf die Unterseite zu verschieben, muss jeder einzelne Vertex um die halbe Höhe des Quaders entlang der entsprechenden Achse verschoben werden:

```
on MovePivotPoint (thisModel, thisVector)    -- this model: Quader, this Vector:
                                                              Verschiebungsvektor

    if not(findPos(thisModel.modifier,#meshDeform))then -- falls Modiefier #meshdeform
                                                                  noch fehlt…
  thisModel.addModifier(#meshDeform)                  -- füge ihn hinzu
 end if

 repeat with meshIndex = 1 to thisModel.meshDeform.mesh.count -- gehe durch
                                                                  sämtliche Meshes

  thisVertexList = thisModel.meshDeform.mesh[meshIndex].vertexList -- schreibe
                                                          alle Veritices in eine Liste
       repeat with vertexIndex = 1 to thisVertexList.count -- gehe durch alle Vertices
   thisVertex = thisVertexList[vertexIndex]
   thisVertex = thisVertex + thisVector          -- addiere Verschiebungsvektorzur Position
     thisVertexList[vertexIndex] = thisVertex    -- schreibe Vertex zurück in Liste
  end repeat
  thisModel.meshDeform.mesh[meshIndex].vertexList = thisVertexList -- schreibe
                                                              Liste zurück in Mesh
 end repeat
end MovePivotPoint
```

Der nahe liegende Gedanke ist, die Rotationsachsen des Quaders, die ja auch in der asf-Datei angegeben werden, einzustellen.
Theoretisch müssen dafür die Positionen sämtlicher Vertices gespeichert werden. Dann kann der Quader um die angegebenen Winkel gedreht werden und anschließend die Vertices unabhängig von den Quaderachsen wieder auf ihre alten Positionen zurückgesetzt werden.

Für diesen Lösungsansatz müssen die Positionen der Vertices in Weltkoordinaten angegeben sein. Leider ist dies in Director nicht der Fall, da alle Positionsangaben von Objektelementen auf die lokalen Achsen bezogen sind und es unmöglich ist, mit Lingo auf entsprechende Weltkoordinaten zuzugreifen.

Um trotzdem die gewünschten Rotationen zu erhalten, wird für jeden Quader ein Dummyobjekt in Form einer Kugel erzeugt. Der Quader wird zu einem Child-Objekt der Kugel. Von nun an folgt er allen Rotationen und Translationen der Kugel. Wird die Kugel in den entsprechenden Winkeln gedreht, richtet sich auch der zugehörende Quader entsprechend aus.

```
on rotateAxis (thisDummy, thisModel, theObj, i)
 axis = theObj.amcAxis(i)
 a = axis[1]
 b = axis[2]
 c = axis[3]

 thisDummy.addChild(thisModel)
 thisDummy.transform.rotation = vector(a, b, c)
end
```

Das nächste Problem entsteht dadurch, dass, im Gegensatz zu einer echten Jointstruktur, die Quader nicht automatisch in die richtige Richtung auf das Parent-Objekt ausgerichtet werden. Wenn alle Quader mit fester Breite und Länge erzeugt werden und die Höhe der angegebenen Bone-Länge gleichgesetzt wird, so wird dies für die Mehrheit der Quader zum gewünschten Ergebnis führen. Aber die Quader der Hüfte (*rhipjoint* und *lhipjoint*) und der Schlüsselbeine (*rclavicle* und *lclavicle*) erscheinen um 90° gedreht. Um diesen Fehler zu beheben, wird bei ihnen die Höhe auf einen festen Wert gesetzt und stattdessen die Breite variiert. Die Vertices werden beim Verschieben des Pivotpunktes nicht entlang der Y-, sondern entlang der X-Achse verschoben.
Die Pivotpunkte der Quader der Wirbelsäule (*lowerback, upperback, thorax, lowerneck, upperneck* und *head*) müssen genau in die entgegengesetze Richtung verschoben werden als das bei allen übrigen der Fall ist.

Um auf diese Eigenheiten eingehen zu können, ist es notwendig, die entsprechenden Bonenamen abzufragen. Dies setzt voraus, dass die beiden Acclaimdateien einer bestimmten Namensgebung und Anordnung der Hierarchie folgen.

```
str = theObj.asfName(i-1)

  if (str.word[1] = "rhipjoint") or \
    (str.word[1] = "lhipjoint") or \
    (str.word[1] = "rclavicle") or \
    (str.word[1] = "lclavicle") then

   BResList[i].width = theObj.asfLength(i-1)*m
   BResList[i].height = 20
  end if
```

Zusätzlich zu Dummykugeln und Bone-Quadern wird noch eine Kugel als Rootbone erzeugt. Während alle anderen Bones nur rotiert werden, sind für den Rootbone auch Positionsdaten abgespeichert. Er ist das Parent-Objekt für die komplette Skeletthierarchie.
Auch für die hierarchische Darstellung werden Boden, Koordinatenursprung und Lichter erzeugt.

Nachdem alle Quader mit ihren entsprechenden Dummyobjekten zusammengefasst und ausgerichtet sind, wird die Hierarchie aufgebaut. Für diesen Zweck wir die Hierarchie-Liste der Indices, die vom Xtra erzeugt wurde, durchlaufen. Wenn der Listeneintrag nicht „-1“ ist, also nicht das Ende eines Hierarchiezweiges erreicht wurde, wird der Eintrag in der Variablen *ParentBone* gespeichert, und der nachfolgende Eintrag in der Variablen *ChildBone*. Als *theParent* wird der Rootbone angegeben, falls *ParentBone* gleich 0 ist. Ist *ParentBone* ungleich 0, wird der entsprechende Quader angegeben.

```
parentBone = hier[counter]
   childBone = hier[counter +1]
   if (parentBone = 0) then
    theParent = member(theStage).model("Root")
   else
    theParent = member(theStage).model("Bone"&parentBone)
   end if
```

Wenn *ChildBone* ungleich „-1“, also ebenfalls ein gültiger Bone ist, so wird die Variable *theChild* auf den entsprechenden Dummy gesetzt.

```
if NOT(childBone = -1) then
    theChild = member(theStage).model("dummy"&childBone)
```

Nun müssen die Child-Objekte auf die richtige Startposition gesetzt werden. Der Positionsvektor berechnet sich aus der Summe der Weltposition des Parents mit dem Produkt des Richtungsvektors des Childs mit seiner Länge.

```
    l = theObj.asfLength(childBone-1)*30/theObj.asfUnit()

   startPos = theParent.worldposition
   dir = theObj.amcDirections(childBone-1)
   vec = vector(dir[1], dir[2], dir[3])

   bonePos = startPos + vec*l

   theChild.transform.position = bonePos
```

Dem Child-Objekt, also der Kugel, die als Dummy-Objekt dient, wird das oben definierte Parent-Objekt zugewiesen.

```
theParent.addChild(theChild,#preserveWorld)
```

Die Abspielschleife ist bei den Rotationsdaten etwas aufwendiger. Der Rootbone kann ganz einfach rotiert und positioniert werden.
Die Rotation der übrigen Bones erfolgt über die Quader, nicht über die Dummyobjekte. Werden die Bones auf die angegebenen Rotationswerte gesetzt, so ist die Rotation zwar richtig, aber die Bones liegen an der falschen Stelle im Raum.

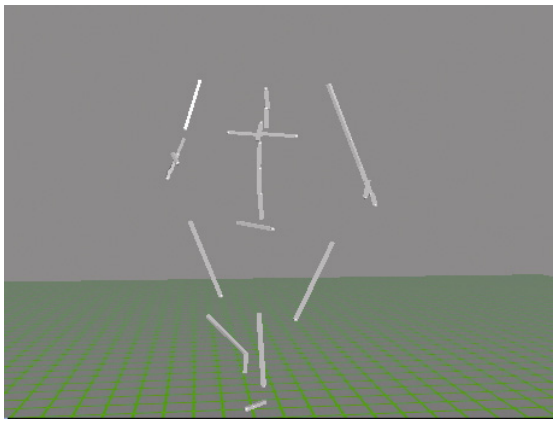

Abbildung 13: Skelett nach der Rotation der Bones

Wird die Position des entsprechenden Dummyobjektes der Position des Parents gleich gesetzt, liegen alle Bones (Quader + Dummy) richtig rotiert an der Position des Root-Bones.

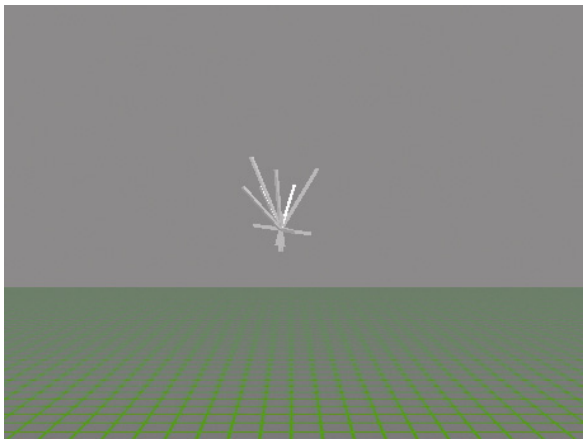

Abbildung 14: Bones nach Verschieben der Dummys

Um das Skelett wieder richtig darzustellen, müssen alle Quader um ihre Länge entlang ihrer Y-Achse (beziehungsweise der X-Achse bei den Hüft- und Schlüsselbein-Bones) verschoben werden. Verschiebt man den Quader selbst, bewegt sich dabei sein zugehörendes Dummy-Objekt, das ja sein Parent ist, nicht mit. Eigentlich müssten die Translationen der Quader über ihre Parent-Objekte erfolgen. Die lokalen Achsen der Dummy-Objekte sind bezüglich der lokalen Achsen der entsprechenden Quader verdreht. Da es nicht möglich ist, die Achsen

eines Quaders als Verschiebungsachsen des Dummys anzugeben, wird die Verschiebung der Dummykugeln entlang der Y-Achse zu einem falschen Ergebnis führen.
Der nahe liegende Gedanke wäre, die Parent-Child-Beziehungen kurzzeitig umzukehren, damit sich der Dummy gemeinsam mit dem Quader bewegt. Theoretisch bleiben die Positionen der Objekte erhalten, doch auch in diesem Fall hält sich die Praxis leider nicht an die Theorie: Die Objekte verschieben sich willkürlich im Raum, sobald sich ihr Parent-Objekt ändert.
Als praktikable Lösung erwies es sich, zuerst den Quader um seine Länge entlang seiner eigenen Achse zu verschieben, wobei sein Dummy-Objekt an Ort und Stelle bleibt. Dann wird die Weltposition des Dummy-Objektes der des Quaders gleich gesetzt, wobei sich der Quader mitbewegt. Darum muss der Quader um seine eigene Länge wieder zurückgeschoben werden.

6.3.2 Ansatz 2: Import eines in Maya erstellten Skeletts im w3d-Format

Nachdem es sich herausgestellt hat, dass eine allgemeine Darstellung des Skeletts nicht möglich ist, führte dies zu der Idee, ein komplettes Skelett in Maya zu kreieren. Mit Hilfe des seit Maya5.0 automatisch zur Verfügung stehenden Shockwave-Exporters kann dies in Director transferiert werden. Die Idee war, mit Hilfe dieses Skelettes, einen komplexen „Actor“, einen menschenähnlichen Darsteller, zur Visualisierung nutzen zu können, statt die Bones einfach nur durch Quader darzustellen.

Mit Hilfe des Modifikators *#bonesplayer* ist es möglich, Bonesanimationen in Director zu manipulieren und zu erstellen. Das heißt, es wäre ohne weiteres möglich, das vordefinierte Skelett in die richtige Ausgangsposition zu bringen.

Leider zeigte es sich sehr schnell, dass es beinahe unmöglich ist, die vordefinierten Rotationen des Mayaskelettes in den Griff zu bekommen. Die Bones über den Bonesplayer anzusprechen und zu verschieben beziehungsweise zu rotieren ist möglich, die Ergebnisse sind allerdings nicht vorhersagbar, da nicht klar ist, wie die Rotationsachsen der Bones bezüglich der Weltkoordinaten ausgerichtet sind.

In beiden Fällen ist es leider unumgänglich, eine festgelegte Skeletthierarchie mit vordefinierten Namen vorauszusetzen. Daher ist eine korrekte Darstellung nur dann möglich, wenn die gespeicherte Skeletthierarchie, der von Vicons BodyBuilder verwendeten Form entspricht.

6.4 Das i3d-Informationsfile

Neben den bereits genannten Dateien, befasst sich 3dCubic auch mit i3d-Dateien. Sie dienen der Speicherung von Daten zu den einzelnen Dateien. Im Idealfall besitzt jede amc-, c3d- und csm-Datei sein eigenes i3d-File. Das i3d-File einer Datei hat immer denselben Namen wie die zugehörige Datei der Marker-, beziehungsweise Rotationsdaten.

Mit Hilfe des FileXtra4 wird die Festplatte nach Dateien des Typs *.c3d, *.amc und *.csm durchsucht. Die gefundenen Dateinamen werden in der Reihenfolge ihres Auftretens in der Lingoliste *fileList* gespeichert. Die dazu gehörenden Pfadnamen werden in einer zweiten Liste *preList* abgelegt. Zusätzlich wird noch eine Liste *i3dList* angelegt. Sie enthält die Namen sämtlicher i3d-Filenamen, unabhängig davon, ob die Datei tatsächlich existiert oder nicht.

```
folders = fObj.fx_FolderToList(drive)
…
repeat with i in folders
    …
    if fObj.fx_FileGetType(drive&i) = ".c3d" then
    append fileList,(i)
    append preList, (drive)
    info = chars(i, 1, (the number of chars in i -4))
    append i3dList, info & ".i3d"
      …
```

Werden die Erweiterten Funktionen aufgerufen, wird mit dem Standard-FileIO-Xtra überprüft, ob die i3d-Datei vorhanden ist. Wenn dies der Fall ist, wird sie ausgelesen, und ihre Werte werden in den Textfeldern neben dem 3D-Darsteller angezeigt. Der Edit- und der Savebutton bleiben in ihren Defaultzuständen: der Editbutton ist auf *Edit* gesetzt und aktiv, der Savbutton ist deaktiviert.

```
 fname = preList[actual]&i3dList[actual]

 if NOT (fObj.fx_fileExists(fname)) then   -- if file does not exist...
  …
 else          -- ...else open it and read it's content
  noI3d = FALSE
    …
  -- prepare fields
  member("tfDate").text = i3dBuffer.line[1].word[2..(the number of words in
i3dBuffer.line[1])]
    …
```

Wenn die Datei nicht existiert, wird der Anwender gefragt, ob er sie jetzt erstellen möchte. Falls er mit „Ja“ antwortet, werden alle bereits bekannten Angaben in die Textfelder eingetragen und alle Textfelder auf editierbar gesetzt. Zur Verdeutlichung werden sie weiß unterlegt. Der Editbutton wird auf *Cancel* gesetzt und der Savebutton wird aktiviert.

```
alertInitList=[#buttons:#YesNo,#title: "Create file", #message:
i3dList[actual]&RETURN&"infofile does not exist."&RETURN&"Wish to
                                                          create?",#icon:#question]
  noI3d = TRUE
  if (alert(muiObj, alertInitList)=1)then  -- ...ask, what to do...
   edit = TRUE
   -- prepare fields
   repeat with i=23 to 33
    num = i
    editField (num)        -- change state from activ to inactiv or the other way round
   end repeat
   -- prepare buttons
   sprite(35).member = member("cancel")
```

```
sprite(35).m_n = member("cancel")
sprite(35).m_hi = member("cancel_S")
sprite(35).m_dn = member("cancel_P")
…
-- insert known data
if(isC3D) then
 member("tfDuration").text = string(c3dData.c3dDuration())
 member("tfFramerate").text = string(c3dData.c3dFrameRate())
 member("tfSubject").text  = string(word 1 of c3dData.c3dSubject())
else if (isCSM) then
 …
```

Möchte der Anwender keine Datei anlegen, wird der Editbutton auf *Create* gesetzt und der Savebutton bleibt inaktiv.

```
edit = FALSE
-- prepare buttons
sprite(35).member = member("create")
sprite(35).m_n = member("create")
sprite(35).m_hi = member("create_S")
sprite(35).m_dn = member("create_P")
```

Sobald der Anwender die Erweiterten Funktionen aufruft, wird der Inhalt der i3d-Datei, wenn sie denn vorhanden ist, in einem String zwischengespeichert.

```
i3d.openFile(fname,1)
i3dBuffer = i3d.readFile()
```

Beschließt der Anwender eine Änderungsaktion abzubrechen, werden die alten Werte wieder in die Textfelder eingetragen. Die i3d-Datei selbst ändert sich erst wenn der Anwender auf *Save* klickt.

Möchte der Anwender speichern, wird zunächst geprüft, ob in der Zeile *Cleaned:* ein gültiger Eintrag[11] vorliegt. Alles andere führen zu der Aufforderung, einen gültigen Eintrag vorzunehmen.

```
if member("tfCleaned").text contains "JA"  or\
    …
    member("tfCleaned").text = EMPTY then
  else
   alertInitList=[#buttons:#OK, #title: "missing data", #message:"Please insert Yes or No
for cleaning status.",#icon:#note]
   alert(muiObj, alertInitList)
    …
  end if
```

Anschließend wird getestet ob es überhaupt möglich ist, in die bestehende Datei zu schreiben, beziehungsweise ob es möglich ist, sie zu erzeugen. Ist dies der Fall, so wird die alte i3d-Datei gelöscht, und anschließend eine neue erzeugt. Die neuen Werte werden aus den Textfeldern ausgelesen und eingetragen.

```
i3d.openFile(fname,0)
i3d.delete()
i3d.createFile(fname)
if i3d.status()=0 then -- everything is allright
 i3d.openFile(fname, 0)

 outText = "Date: "&member("tfDate").text&RETURN&\
   "Group: "&member("tfGroup").text&RETURN&\
   …
 i3d.writeString(outText)
 i3d.closeFile()
```

Andernfalls wird die Aktion wird abgebrochen und der Anwender darüber informiert.

```
    else
     alertInitList=[#buttons:#OK, #title: "abort writing", #message:"File could not be open
for writing",#icon:#note]
     alert(muiObj, alertInitList)
   end if
```

[11] Gültige Einträge sind: *J, Ja, Yes, Y, N, Nein* oder *No* (Anmerkung des Autors)

7 Mögliche Verbesserungen

In diesem Kapitel werden Vorschläge für mögliche Weiterführungen und Verbesserungen von 3DCubic in einer zweiten Version aufgezeigt.

7.1 Repräsentation der Marker durch einen "Actor"

In der vorliegenden Version der Anwendung 3DCubic ist die Darstellung der Markerinformationen nur möglich, wenn die Markerinformationen direkt in Form von Kugeln, die den tatsächlich aufgenommenen Markern entsprechen, dargestellt werden. Bei komplexen Bewegungen ist es daher manchmal sehr schwierig, die genaue Bewegung zu erkennen, da nicht mehr eindeutig erkennbar ist, welche Marker welchem Körperteil zugeordnet sind. Die Darstellung der Bewegungen des aufgenommenen Darstellers wäre leichter interpretierbar, wenn diese Kugeln wenigstens durch Linien miteinander verbunden wären, um ein leichteres Erkennen der tatsächlichen Bewegung zu ermöglichen.

In der aktuellen Version wird von überwiegend ungecleanten Daten ausgegangen. Das bedeutet, manche Marker sind wegen großer Lücken innerhalb einer Bewegungsbahn bei den Aufzeichnungen mehrfach vorhanden, da die Workstation bei der Aufnahme nicht in der Lage war, sie richtig zuzuordnen. Statt der 41 Marker, die Vicon für die Bewegungsaufnahme eines Menschen eigentlich vorgesehen hat, werden häufig deutlich mehr Marker aufgelistet. Insbesondere die Marker des Brustbeins und der Schlüsselbeine, sowie die vorderen Marker an der Hüfte, werden oft über einen längeren Zeitraum verdeckt und anschließend als neue Marker gewertet. Dieser Umstand macht es sehr schwierig, Director anzuweisen, zwischen bestimmten Markern Verbindungslinien darzustellen, da das eine Ende der Verbindungslinie plötzlich durch einen anderen Marker repräsentiert werden kann. Daher wurde auf in der vorliegenden Version von 3DCubic auf ihre Darstellung verzichtet.

Sind die Daten gesäubert, so sind die Bewegungslinien der Marker, ihre Trajektorien, zu einer einzigen Trajektorie zusammengefasst, wenn erkennbar war, dass es sich um denselben Marker handelt. Das Markerset besteht somit aus einer festen Anzahl von Markern und es ist immer eindeutig nachvollziehbar, welche Marker zu einander in Beziehung gesetzt werden können. Dies ermöglicht die Darstellung der Verbindungslinien benachbarter Marker, da ihre Endpunkte über die ganze Sequenz hinweg klar definiert sind.

Die vorliegende Anwendung achtet bereits darauf, dass der *Cleaned*-Eintrag in das i3d-File eine bestimmte Form haben muss. Dadurch soll ermöglicht werden,

dass eine Abfrage erfolgen kann, ob ein „sauberes“ Datenfile vorliegt. In diesem Fall sollte dem Anwender die Möglichkeit gegeben werden, sich Verbindungslinien anzeigen zu lassen.

Als zusätzliche Option sollte es möglich sein, ähnlich wie bei den Acclaim-Dateien, die Position der Gelenke zu berechnen und somit die Darstellung durch einen „Actor“ zu ermöglichen, der grob menschliche Formen aufweist.
Dies ist möglich, wenn jeder Quader, der einem bestimmten Körperteil zugeordnet ist, durch ein passendes Modell ersetzt wird. So wird der Headbone beispielsweise durch eine langgezogene Kugel oder der Thoraxbone durch ein Objekt ersetzt, dass dem menschlichen Brustkorb nachgebildet ist.

7.2 Schließen der Informationslücken durch Interpolation

Selbst wenn die richtige Anzahl an Markern vorliegt, kann es zu Schwierigkeiten kommen: Manchmal werden für einen Marker über mehrere Frames hinweg keine Positionsdaten erfasst. Dies führt bei der vorliegenden Darstellung dazu, dass die Position des entsprechenden Markers auf den Ursprung und seine Sichtbarkeit auf *#none* gesetzt wird. Dies hat aber zur Folge, dass unter Umständen Teile des Markersets über einen längeren Zeitraum nicht dargestellt werden und über mehrere Frames hinweg beispielsweise die Marker des Brustbeins oder der Hüfte fehlen.

Um diese Lücken in der Darstellung abzufangen, müssen die fehlenden Positionsdaten berechnet werden.

Lingo bietet von sich aus zwei Kommandos, um Positionsdaten interpolieren zu lassen:

> *„Die Syntax für das Kommando interpolate ist:*
>
> *newTransform = startTransform.interpolate(endTransform, percent)*
>
> *Der percent-Wert ist eine Fließkommazahl und 50% werden als 50 und nicht als 0.5 angegeben. Die Transformation newTransform ist der Wert, der zurückgeliefert wird. Die Transformationen startTransform und endTransform werden nicht verändert*
>
> *Die Syntax für das interpolateTo-Kommando ist:*
>
> *startTransform.interpolateTo(endTransform, percent)*
>
> *Mit dem interpolateTo-Kommando wird die startTransfromation verändert, statt dass eine neue Transformation erstellt wird.“*[12]

[12] *Gross, Phil und Mike,* Director 8.5, Shockwavestudio für 3D, 2002, S.413

Werden diese Interpolationskommandos sinnvoll eingesetzt, sollte es möglich sein, größere Lücken in den Positionsdaten der Marker zu schließen. Probleme ergeben sich dann, wenn der Schauspieler plötzlich seine Bewegungsrichtung geändert hat, während Marker verdeckt waren. Das ist vor allem bei Lücken, die über mehrere Sekunden reichen, sehr wahrscheinlich. Wirklich gut funktioniert die Interpolation nur dann, wenn die fehlende Bewegungskurve des Markers mehr oder minder linear verlaufen wäre.

Bei Dateien, die noch zu viele Trajektorien enthalten, muss auch geprüft werden, ob nach dem letzten gültigen Wert überhaupt noch ein sinnvoller Wert folgt. Andernfalls darf auf keinen Fall interpoliert werden, da sonst alle Marker, deren Bewegungsbahnen zu irgendeinem Zeitpunkt nicht mehr weitergeführt wurden, von ihrer letzten Position aus langsam zum Ursprung wandern.

7.3 Anbindung an eine Datenbank

Die i3d-Files bieten die Möglichkeit, zusätzliche Informationen zu den Dateien in einem vordefinierten Format abzulegen.

Eine weitere Verbesserung wäre es, wenn die Anwendung neben dem Durchsuchen der Festplatte auch noch die Möglichkeit bieten würde, eine Datenbank nach Einträgen wie zum Beispiel Darstellernamen, Aufnahmedatum oder Art der Bewegung zu durchsuchen. Somit wäre es beispielsweise möglich, alle 3D-Files anzeigen zu lassen, die mit einem bestimmten Darsteller aufgezeichnet wurden.

Die gefundenen Ergebnisse könnten dann in über die visuelle Darstellung von 3DCubic gleich nach ihrer Verwendbarkeit überprüft werden.

Für diesen Zweck sollte 3DCubic zusätzlich so erweitert werden, dass der Anwender die Möglichkeit hat, neue Dateien über 3DCubic in die bestehende Datenbank einzupflegen.

8. Zusammenfassung und Ausblick

Es hat sich gezeigt, dass Director für die Realisation dieses Projektes sehr gut geeignet war. Insbesondere durch die Erweiterbarkeit durch Xtras und durch die gut durchdachte Skriptsprache Lingo ist Director ein sehr gutes Werkzeug, um interaktive Anwendungen in Kombination mit echter Dreidimensionalität umzusetzen. Vor allem überraschte die gute Zugriffsmöglichkeit auf alle Elemente innerhalb der 3D-Bühne, wie beispielsweise Lichter, Kameras, Texturen und die Objekte selbst.

Die Zielsetzung, die oben genannten Dateitypen in einem 3D-Raum darzustellen, wurde erreicht. Das entstandene Tool 3DCubic kann als Sichtungs- und Verwaltungsprogramm für Motion Capture-Dateien von Nutzen sein.

An einigen Stellen weißt die aktuelle Version leichte Schwächen in der Performance auf, wenn sie auf älteren Rechnern läuft. Aber auf den Arbeitsrechnern, wie sie im CA-Labor der HdM bereitstehen, und auf denen 3DCubic wohl vor allem eingesetzt werden wird, fallen diese nicht mehr ins Gewicht.

Vielleicht werden sich in Zukunft noch weitere 3D-Formate etablieren oder weitere Formen der Darstellung von Daten gewünscht. Da Director stets weiterentwickelt wird, wird es in zukünftigen Versionen von 3DCubic möglich sein, auf alle neuen Anforderungen einzugehen.

Anhang A: Quellcodes der Xtras

A.1 Das c3dXtra

Neben den selbst geschriebenen Dateien c3dData.h und c3dData.cpp sind in diesem Unterkapitel auch die Dateien des MOA-Script-Skeletts dargestellt, die zur Erstellung des Xtras geändert wurden. Das sind im Einzelnen: CRegster.h, CScript.h, CRegster.cpp und CScript.cpp. Alle Änderungen sind in rot hervorgehoben. In den Unterkapiteln A.2 und A.3 wird auf die erneute Darstellung dieser Dateien verzichtet.

CRegster.h

```
#ifndef _H_cregster

#define _H_cregster

/*****************************************************************************
 * INCLUDE FILE(S)
 *****************************************************************************/
#ifndef _H_moaxtra
   #include "moaxtra.h"
#endif

#ifndef _H_cscript
   #include "cscript.h"
#endif

#ifndef _Hc3dData
   #include "c3dData.h"
#endif

/*****************************************************************************
 * CONFIGURATION DEFINE(S)
 * -----------------------
 * Uncomment the line below to activate debugging or load control code.
 *****************************************************************************/
//#define MOA_DEBUG
//#define USING_LOAD_CONTROL

/*****************************************************************************
 * CLASS SPECIFIC DEFINE(S)
 *****************************************************************************/
#ifndef UNUSED
   #define UNUSED(x) x
#endif

/*****************************************************************************
 * CLSID
 *****************************************************************************/
// {2C065ACE-6994-4af0-89F4-7A66CDE69BD6}
DEFINE_GUID(CLSID(CRegister),
0x2c065ace, 0x6994, 0x4af0, 0x89, 0xf4, 0x7a, 0x66, 0xcd, 0xe6, 0x9b, 0xd6);


/*****************************************************************************
 * CLASS INSTANCE VARIABLES
 *****************************************************************************/
EXTERN_BEGIN_DEFINE_CLASS_INSTANCE_VARS(CRegister)
   /*
    * ---> insert additional variable(s) -->
    */
EXTERN_END_DEFINE_CLASS_INSTANCE_VARS

/*****************************************************************************
 * CLASS INTERFACE(S)
 *****************************************************************************/
EXTERN_BEGIN_DEFINE_CLASS_INTERFACE(CRegister, IMoaRegister)
   EXTERN_DEFINE_METHOD(MoaError, Register, (PIMoaCache, PIMoaDict))
EXTERN_END_DEFINE_CLASS_INTERFACE

/*
 * --> insert additional interface(s) -->
 */
```

```
#endif /* _H_cregster */
```

CScript.h

```
#ifndef _H_cscript
#define _H_cscript

/*****************************************************************************
 * INCLUDE FILE(S)
 *****************************************************************************/
#ifndef _H_moaxtra
   #include "moaxtra.h"
#endif

#ifndef _H_mmiservc
   #include "mmiservc.h"
#endif

#ifndef _H_mmixscrp
   #include "mmixscrp.h"
#endif

#ifndef _H_driservc
   #include "driservc.h"
#endif

#ifdef MACINTOSH
   #include <windows.h>
   #include <string.h>
#endif

/*****************************************************************************
 * CONFIGURATION DEFINE(S)
 * -----------------------
 * Uncomment any appropriate lines below to implement the indicated Xtra
 * interfaces within this class.
 *****************************************************************************/
//#define USING_INIT_FROM_DICT
//#define USING_NOTIFICATION_CLIENT

/*****************************************************************************
 * CLASS SPECIFIC DEFINE(S)
 *****************************************************************************/
#ifndef UNUSED
   #define UNUSED(x) x
#endif

/*****************************************************************************
 * CLSID
 *****************************************************************************/
// {46E4D2E4-160B-4af0-9BC7-BCD0061E0624}
DEFINE_GUID(CLSID(CScript),
0x46e4d2e4, 0x160b, 0x4af0, 0x9b, 0xc7, 0xbc, 0xd0, 0x6, 0x1e, 0x6, 0x24);


/*****************************************************************************
 * CLASS INSTANCE VARIABLES
 *****************************************************************************/
EXTERN_BEGIN_DEFINE_CLASS_INSTANCE_VARS(CScript)
   PIMoaMmValue pValue;
   PIMoaMmList pList;
   PIMoaMmUtils2 pMmUtils;
   PIMoaDrPlayer pDrPlayer;
EXTERN_END_DEFINE_CLASS_INSTANCE_VARS

/*****************************************************************************
 * CLASS INTERFACE(S)
 *****************************************************************************/
EXTERN_BEGIN_DEFINE_CLASS_INTERFACE(CScript, IMoaMmXScript)
   EXTERN_DEFINE_METHOD(MoaError, Call, (PMoaMmCallInfo))

  private:
   EXTERN_DEFINE_METHOD(MoaError, XScrpC3dOpen,       (PMoaDrCallInfo callPtr))
   EXTERN_DEFINE_METHOD(MoaError, XScrpC3dGetData,    (PMoaDrCallInfo callPtr))
   EXTERN_DEFINE_METHOD(MoaError, XScrpC3dDuration,   (PMoaDrCallInfo callPtr))
   EXTERN_DEFINE_METHOD(MoaError, XScrpC3dNTrajectories, (PMoaDrCallInfo callPtr))
   EXTERN_DEFINE_METHOD(MoaError, XScrpC3dSubject,    (PMoaDrCallInfo callPtr))
   EXTERN_DEFINE_METHOD(MoaError, XScrpC3dFrameRate,  (PMoaDrCallInfo callPtr))

EXTERN_END_DEFINE_CLASS_INTERFACE

#ifdef USING_INIT_FROM_DICT
EXTERN_BEGIN_DEFINE_CLASS_INTERFACE(CScript, IMoaInitFromDict)
   EXTERN_DEFINE_METHOD(MoaError, InitFromDict, (PIMoaRegistryEntryDict))
EXTERN_END_DEFINE_CLASS_INTERFACE
#endif

#ifdef USING_NOTIFICATION_CLIENT
EXTERN_BEGIN_DEFINE_CLASS_INTERFACE(CScript, IMoaNotificationClient)
   EXTERN_DEFINE_METHOD(MoaError, Notify, (ConstPMoaNotifyID, PMoaVoid, PMoaVoid))
EXTERN_END_DEFINE_CLASS_INTERFACE
#endif
```

```
enum
{
    m_new = 0,                  /* standard */

    m_c3dOpen,
    m_c3dGetData,
    m_c3dDuration,
    m_c3dNTrajectories,
    m_c3dSubject,
    m_c3dFrameRate,

    m_XXXX
};

#endif /* _H_cscript */
```

CRegster.cpp

```
/************************************************************************
 * INCLUDE FILE(S)
 ************************************************************************/
#ifndef INITGUID    /* INITGUID causes the actual GUIDs to be declared, */
    #define INITGUID   /* not just references to them. Must be declared in */
#endif         /* only one file. */

#ifndef _H_cregster
    #include "cregster.h"
#endif


/************************************************************************
 * CLASS INTERFACE(S)
 ************************************************************************/
BEGIN_DEFINE_CLASS_INTERFACE(CRegister, IMoaRegister)
END_DEFINE_CLASS_INTERFACE


static char msgTable[] = {
    "xtra c3dReader\n" \
    "new object me\n" /* standard first handler entry in all message tables */
    "-- Template handlers --\n"
    "c3dOpen object, string filename \t-- opens a c3d-file \n"
    "c3dGetData object \t\t-- return x-, y- and z-data\n"
    "c3dDuration object me \t\t-- returns duration\n"
    "c3dNTrajectories object me\t-- returns number of trajectories \n"
    "c3dSubject object me\t\t-- returns name of captured subject\n"
    "c3dFrameRate object me \t\t-- returns used frame rate\n"
    };

/************************************************************************
 * CREATE AND DESTROY METHODS
 ************************************************************************/

/* ------------------------------------------------- MoaCreate_CRegister */
STDMETHODIMP MoaCreate_CRegister(CRegister FAR * This)
{
    UNUSED(This);

    MoaError     err = kMoaErr_NoErr;

    /*
     * --> insert additional code -->
     */

    return(err);
}

/* ------------------------------------------------ MoaDestroy_CRegister */
STDMETHODIMP_(void) MoaDestroy_CRegister(CRegister FAR * This)
{
    UNUSED(This);

    /*
     * --> insert additional code -->
     */

     return;
}

/* --------------------------------- CRegister_IMoaRegister Create/Destroy */
CRegister_IMoaRegister::CRegister_IMoaRegister(MoaError FAR * pErr)
    { *pErr = (kMoaErr_NoErr); }
CRegister_IMoaRegister::~CRegister_IMoaRegister() {}

/*
 * --> insert additional create/destroy method(s) -->
 */

/************************************************************************
 * METHOD IMPLEMENTATION(S)
 ************************************************************************/
```

```
/* ------------------------------------------ CRegister_IMoaRegister::Register */
STDMETHODIMP_(MoaError) CRegister_IMoaRegister::Register(PIMoaCache pCache,
    PIMoaDict pXtraDict)
{

    /* variable declaration */
    MoaError  err = kMoaErr_NoErr;
    PIMoaDict pRegDict;

    #ifdef USING_LOAD_CONTROL
    MoaLong       loadOptions = (kMoaXlc_PreloadAtLaunch | kMoaXlc_NeverUnload);
    err = pCache->AddRegistryEntry(pXtraDict, &CLSID_CInterrogate, &IID_IMoaXtraLoadControl, &pRegDict);
    err = pRegDict->Put(kMoaRegType_XtraLoadControlOptions, &loadOptions, 0,
kMoaRegKey_XtraLoadControlOptions);
    #endif

    /* Register the scripting xtra */
    err = pCache->AddRegistryEntry(pXtraDict, &CLSID_CScript, &IID_IMoaMmXScript, &pRegDict);

    if ( err == kMoaErr_NoErr)
    {
        /* Register the method table */
        err = pRegDict->Put(kMoaMmDictType_MessageTable, msgTable, 0, kMoaMmDictKey_MessageTable);
    }

    #define kMoaMmDictKey_SafeForShockwave "safeForShockwave"
    #define kMoaMmDictType_SaveForShockwave kMoaDictType_Bool
    MoaBool bItsSafe = TRUE;
    if(err == kMoaErr_NoErr)
    {
        err = pRegDict->Put(kMoaMmDictType_SaveForShockwave, &bItsSafe, sizeof(bItsSafe),
kMoaMmDictKey_SafeForShockwave);
    }
    return(err);
}
```

CScript.cpp

```
/*************************************************************************
 * INCLUDE FILE(S)
 *************************************************************************/
#ifndef _H_cscript
    #include "cscript.h"
#endif

#ifdef MACINTOSH
    #include <windows.h>
    #include <string.h>
#endif

#ifndef _H_c3dData
    #include "c3dData.h"
#endif

/*************************************************************************
 * Private Methods
 *************************************************************************/

/* ------------------------------------------------- XScrpc3dOpen */
#define ARG_BASE 1
c3dData *data;
MoaError CScript_IMoaMmXScript::XScrpC3dOpen(PMoaDrCallInfo callPtr)
{
    MoaMmValue value;
    MoaChar filename[256] = {0};
    MoaError err = kMoaErr_NoErr;

    AccessArgByIndex(ARG_BASE + 1, &value);
    pObj->pValue->ValueToString(&value,&filename[0],sizeof(filename));

    data = new c3dData;

    data->openFile(filename);

    return(err);
}

/*------------------------------------------------------- XScrpc3dGetData*/
MoaError CScript_IMoaMmXScript::XScrpC3dGetData(PMoaDrCallInfo callPtr)
{
    MoaMmValue value, data3d;
    MoaError err = kMoaErr_NoErr;


    pObj->pList->NewListValue(&data3d);

    for (int i = 0; i< data->Data.size(); i++)
    {
        pObj->pValue->FloatToValue(data->Data[i], &value);
        pObj->pList->AppendValueToList(&data3d, &value);
    }
    callPtr->resultValue = data3d;
```

```
    return(err);
}


/* ------------------------------------------------------ XScrpc3dDuration */
MoaError CScript_IMoaMmXScript::XScrpC3dDuration(PMoaDrCallInfo callPtr)
{
    MoaMmValue dur;
    MoaError err = kMoaErr_NoErr;

    pObj->pValue->IntegerToValue(data->duration, &dur);
    callPtr->resultValue = dur;
    return(err);
}

/* ------------------------------------------------------ XScrpc3dNTrajectories */
MoaError CScript_IMoaMmXScript::XScrpC3dNTrajectories(PMoaDrCallInfo callPtr)
{
    MoaMmValue nTraj;
    MoaError err = kMoaErr_NoErr;

    pObj->pValue->IntegerToValue(data->numTraj, &nTraj);
    callPtr->resultValue = nTraj;
    return(err);
}


/* ------------------------------------------------------ XScrpc3dSubject */
MoaError CScript_IMoaMmXScript::XScrpC3dSubject(PMoaDrCallInfo callPtr)
{
    MoaMmValue name;
    MoaError err = kMoaErr_NoErr;

    pObj->pValue->StringToValue(data->name, &name);
    callPtr->resultValue = name;
    return(err);
}

/* ------------------------------------------------------ XScrpc3dFrameRate */
MoaError CScript_IMoaMmXScript::XScrpC3dFrameRate(PMoaDrCallInfo callPtr)
{
    MoaError err = kMoaErr_NoErr;
    MoaMmValue sRate;
    pObj->pValue->FloatToValue(data->sampleRate, &sRate);
    callPtr->resultValue = sRate;
    return(err);
}



/*****************************************************************************
 * CLASS INTERFACE(S)
 *****************************************************************************/
BEGIN_DEFINE_CLASS_INTERFACE(CScript, IMoaMmXScript)
END_DEFINE_CLASS_INTERFACE

#ifdef USING_INIT_FROM_DICT
BEGIN_DEFINE_CLASS_INTERFACE(CScript, IMoaInitFromDict)
END_DEFINE_CLASS_INTERFACE
#endif

#ifdef USING_NOTIFICATION_CLIENT
BEGIN_DEFINE_CLASS_INTERFACE(CScript, IMoaNotificationClient)
END_DEFINE_CLASS_INTERFACE
#endif

/*
 * --> insert additional method(s) -->
 */

/*****************************************************************************
 * CREATE AND DESTROY METHODS
 *****************************************************************************/

/* ------------------------------------------------------ MoaCreate_CScript */


STDMETHODIMP MoaCreate_CScript(CScript FAR * This)
{
    UNUSED(This);
    /* variable declarations */
    MoaError err = kMoaErr_NoErr;

    err = This->pCallback->QueryInterface(&IID_IMoaMmValue, (PPMoaVoid)&This->pValue);
    err = This->pCallback->QueryInterface(&IID_IMoaMmList, (PPMoaVoid)&This->pList);

    err = This->pCallback->QueryInterface(&IID_IMoaMmUtils2, (PPMoaVoid)&This->pMmUtils);
    err = This->pCallback->QueryInterface(&IID_IMoaDrPlayer, (PPMoaVoid)&This->pDrPlayer);

    return(err);
}

/* ------------------------------------------------------ MoaDestroy_CScript */
STDMETHODIMP_(void) MoaDestroy_CScript(CScript FAR * This)
{
    UNUSED(This);
    if(This->pValue)
```

```
    {
        This->pValue->Release();
        This->pValue=NULL;
    }

    if(This->pList)
    {
        This->pList->Release();
        This->pList=NULL;
    }

    if(This->pMmUtils)
    {
        This->pMmUtils->Release();
        This->pMmUtils=NULL;
    }

    if(This->pDrPlayer)
    {
        This->pDrPlayer->Release();
        This->pDrPlayer=NULL;
    }

    return;
}

/* ---------------------------------- CScript_IMoaMmXScript Create/Destroy */
CScript_IMoaMmXScript::CScript_IMoaMmXScript(MoaError FAR * pErr)
    { *pErr = (kMoaErr_NoErr); }
CScript_IMoaMmXScript::~CScript_IMoaMmXScript() {}

#ifdef USING_INIT_FROM_DICT
/* ------------------------------- CScript_IMoaInitFromDict Create/Destroy */
CScript_IMoaInitFromDict::CScript_IMoaInitFromDict(MoaError FAR * pErr)
    { *pErr = (kMoaErr_NoErr); }
CScript_IMoaInitFromDict::~CScript_IMoaInitFromDict() {}
#endif

#ifdef USING_NOTIFICATION_CLIENT
/* ------------------------- CScript_IMoaNotificationClient Create/Destroy */
CScript_IMoaNotificationClient::CScript_IMoaNotificationClient(MoaError FAR * pErr)
    { *pErr = (kMoaErr_NoErr); }
CScript_IMoaNotificationClient::~CScript_IMoaNotificationClient() {}
#endif

/*
 * --> insert additional create/destroy method(s) -->
 */

/*****************************************************************************
 * METHOD IMPLEMENTATION(S)
 *****************************************************************************/

/* ------------------------------------------- CScript_IMoaMmXScript::Call */
STDMETHODIMP CScript_IMoaMmXScript::Call(PMoaMmCallInfo callPtr)
{

    /* variable declarations */
    MoaError err = kMoaErr_NoErr;

    switch ( callPtr->methodSelector )
    {
        case m_new:
            break;

        case m_c3dOpen:
            err = XScrpC3dOpen(callPtr);
            break;
        case m_c3dGetData:
            err = XScrpC3dGetData(callPtr);
            break;
        case m_c3dDuration:
            err = XScrpC3dDuration(callPtr);
            break;
        case m_c3dNTrajectories:
            err = XScrpC3dNTrajectories(callPtr);
            break;
        case m_c3dSubject:
            err = XScrpC3dSubject(callPtr);
            break;
        case m_c3dFrameRate:
            err = XScrpC3dFrameRate(callPtr);
            break;

        break;
    }

    return(err);
}

#ifdef USING_INIT_FROM_DICT
/* ------------------------------- CScript_IMoaInitFromDict::InitFromDict */
```

```
STDMETHODIMP CScript_IMoaInitFromDict::InitFromDict(PIMoaRegistryEntryDict pRegistryDict)
{
    UNUSED(pRegistryDict);

    /* variable declarations */
    MoaError err = kMoaErr_NoErr;

    /*
     * --> insert additional code -->
     */

    return(err);
}
#endif

#ifdef USING_NOTIFICATION_CLIENT
/* ------------------------------- CScript_IMoaNotificationClient::Notify */
STDMETHODIMP CScript_IMoaNotificationClient::Notify(ConstPMoaNotifyID nid, PMoaVoid pNData, PMoaVoid
pRefCon)
{
    UNUSED(nid);
    UNUSED(pNData);
    UNUSED(pRefCon);

    /* variable declarations */
    MoaError err = kMoaErr_NoErr;

    /*
     * --> insert additional code -->
     */

    return(err);
}
#endif
```

c3dData.h

```
#ifndef _H_c3dData
#define _H_c3dData
#include <vector>
#include <string>
using namespace std;

class c3dData
{
/*****************************************************************************
 * Public Methods Prototypes
 *****************************************************************************/
   public:
       int openFile (char* filename);
       c3dData();
       int readHeader (void);
       void readParameterSection(void);
       void readGroup(int l, int id);
       void readParameter(int l, int id);
       void getParameterData (int type, int nDim, int* dims);
       void readDataSection(void);
       void fillLabels(char* data, int nSigns, int lLabel);

/*****************************************************************************
 * Public Variables
 *****************************************************************************/

       int paramPos;            //Nummer des 512-Byte-Blocks, in dem die Parametersektion beginnt
       int dataPos;             //Nummer des 512-Byte-Blocks, in dem die Datensektion beginnt
       int numTraj;             //Anzahl Trajektorien
       int numAnalog;           //Anzahl der Analogkanaele
       int duration;            //Dauer in Frames
       int processorType;       //Prozessortyp: 84=Intel, 85=DEC (VAX, PDP-11), 86=MIPS (SGI)
       int ID;                  //Aktuelle ID Nummer, zur Überprüfung von Daten

       float sampleRate;
       float scaleFactor;       //Skalierfaktor, falls 3D-Daten als Integer gespeichert sind, müssen sie
                                damit multipliziert werden
       bool flagFP;             //True, wenn 3D-Daten als Floating-Point vorliegen
       bool inParamSection;     //True, soleange innerhalb der ParameterSektion
       bool subNames;           //True, Wenn in Gruppe SUBJECT Parametername NAMES gelesen wurde
       char *name;
       vector<float> Data;      //3D-Koordinaten der Daten-Sektion frame- und trajectorienweise X-,Y- und
                                Z-Koordinate
       FILE *c3dfile;
};

c3dData::c3dData()
{
    paramPos=0;
    dataPos=0;
    numTraj=0;
    numAnalog=0;
    duration=0;
    processorType=0;
```

```
    ID=0;
    sampleRate=0.0;
    scaleFactor=0.0;
    flagFP=false;
    inParamSection=false;
    markersOut=false;
}

#endif /*_H_c3dData */
```

c3dData.cpp

```
#ifndef _H_c3dData
    #include "c3dData.h"
#endif

int c3dData::openFile(char* filename)
{
    c3dfile = fopen(filename, "rb");

    if(!c3dfile) return 0;
    if(!readHeader ()) return 0;

    int seek = fseek(c3dfile,(paramPos-1)*512+2, SEEK_SET);   //Setze File_Zeiger auf Start der
                                                               Parametersektion
    unsigned char c[2];

    fread(c, sizeof(char), 2, c3dfile);                 //Einlesen von 2*reserviertes Zeichen, Anzahl der
                                                        Blöcke der Parametersektion, Prozessortyp
    processorType = c[1];
    inParamSection = true;

    readParameterSection();

    seek = fseek(c3dfile,(dataPos-1)*512, SEEK_SET);    //Setze File_Zeiger auf Start der Datensektion
    readDataSection();

    fclose(c3dfile);

    return 1;
}

int c3dData::readHeader ()
{
    unsigned char c[3];
    unsigned short i[10];
    float f[2];

    fread(c,sizeof(unsigned char),2 ,c3dfile); //Einlesen: Start der ParameterSektion und c3d-File-
                                               Prüfzahl (80)
    if(c[1] == 80)                              //Prüfen, ob wirklich c3d-File vorliegt
    {

        fread(i,sizeof(short),5 ,c3dfile);      //Einlesen: Anzahl Trajectorien, Anzahl Analogwerte/3D-
                                                Frame, Nummer 1. Frame, Nummer letztes Frame, maxilmale
                                                Interpolationslücke
        fread(f, sizeof(float), 1, c3dfile);    //Einlesen: Skalierungsfaktor
        fread(i+6, sizeof (short), 2, c3dfile); //Einlesen: Start der Datensektion
        fread(f+1, sizeof(float), 1, c3dfile);  //Einlesen: Abtastrate [Hz]

        paramPos = (int)c[0];
        dataPos = i[6];
        numTraj = i[0];
        numAnalog = i[1];
        duration = i[3]-i[2]+1;
        sampleRate = f[1];
        if(f[0] < 0) flagFP = true;       //Überprüfen, ob die 3D-Daten als float oder int vorliegen
        else
        {
            flagFP = false;
            scaleFactor = f[0];           //falls Daten als int vorliegen: Skalierfaktor speichern
        }
        return 1;
    }
    else return 0;                        //kein c3d-File
}

void c3dData::readParameterSection()
{
    unsigned char c[3];
    fread(c, sizeof(char), 2, c3dfile);            //Einlesen Länge des Namens, Gruppen-/Parameter-ID

    int nameLength = 0;
    nameLength = c[0];

    if (nameLength == 0) inParamSection = false;    //Wenn nameLength == 0 ist das Ende der
                                                    ParameterSektion erreicht
    while (inParamSection)
    {
```

```
        ID = c[1];
        if(ID > 127) readGroup(nameLength, ID);     //Wenn ID > 127, folgt Gruppe,...
        else readParameter(nameLength, ID);         //... sonst Parameter
        readParameterSection();
    }
}


void c3dData::readGroup(int nameL, int id)
{
    char groupName[256];
    char description[512];
    short offset[1];
    int lDescr = 0;
    id -=256;

    fread(groupName, sizeof(char), nameL, c3dfile);    //Einlesen Gruppenname
    for(int i=0; i<nameL; i++)
    printf("\nID: %d\n", id);
    fread(offset, sizeof(short), 1, c3dfile);       //Einlesen Offset zur nächsten Gruppe/zum nächsten
                                                    Parameter in Byte
    fread(description, sizeof(char), 1, c3dfile);   //Einlesen Länge der Gruppenbeschreibung in Byte
    lDescr = (int)description[0];
    fread(description,sizeof(char), lDescr, c3dfile);  //Einlesen der Gruppenbeschreibung
}

void c3dData::readParameter(int nameL, int id)
{
    char paramName[256];
    unsigned char lDescription[1];
    char description[512];
    short offset;
    int lDescr = 0;
    unsigned char dataType[1];
    unsigned char nDimensions[1];
    int nDim = 0;
    int type = 0;
    int i=0;

    fread(paramName, sizeof(char), nameL, c3dfile);    //Einlesen Parametername
    printf("\nID: %d\n\n", ID);
    fread(&offset, sizeof(short), 1, c3dfile);  //Einlesen Offset zur nächsten Gruppe/zum nächsten
                                                Parameter in Byte
    fread(dataType, sizeof(char), 1, c3dfile);//Einlesen der Groesse des abgelegten Datentyps in Byte
    type = dataType[0];

    fread(nDimensions, sizeof(char), 1, c3dfile);      //Einlesen Anzahl Dimensionen
    nDim = nDimensions[0];
    char comp[] = ("NAMES");

    int vgl2 = strncmp(comp, paramName, 5);  //Überprüfen, ob gerade Gruppe Subject Parameter Names
                                             gelesen wurde, falls ja...
    if(vgl2==0 && ID == 2)
        subNames = true;                     //...subNames = true
    else
        subNames = false;                    //...sonst subNames = false

    int dimensions[7];

    switch (nDim)                            //Fallunterscheidung nach Dimensionenanzahl
    {

    case 0:                                          //Einlesen der einzelnen Dimensionsgrößen
        getParameterData(type, nDim, dimensions);
        break;
    case 1:
        unsigned char d1[1];
        fread(d1, sizeof(char), nDim, c3dfile);
        dimensions[0] = (int)d1[0];
        getParameterData(type, nDim, dimensions);
        break;
    case 2:
        unsigned char d2[2];
        fread(d2, sizeof(char), nDim, c3dfile);
        for(i= 0; i < nDim; i++) dimensions[i] = (int)d2[i];
        getParameterData(type, nDim, dimensions);
        break;
    case 3:
        unsigned char d3[3];
        fread(d3, sizeof(char), nDim, c3dfile);
        for(i= 0; i < 3; i++) dimensions[i] = (int)d3[i];
        getParameterData(type, nDim, dimensions);
        break;
    case 4:
        unsigned char d4[4];
        fread(d4, sizeof(char), nDim, c3dfile);
        for(i= 0; i < nDim; i++) dimensions[i] = (int)d4[i];
        getParameterData(type, nDim, dimensions);
        break;
    case 5:
        unsigned char d5[5];
        fread(d5, sizeof(char), nDim, c3dfile);
```

```
        for(i= 0; i < nDim; i++) dimensions[i] = (int)d5[i];
        getParameterData(type, nDim, dimensions);
        break;
    case 6:
        unsigned char d6[6];
        fread(d6, sizeof(char), nDim, c3dfile);
        for(i= 0; i < nDim; i++) dimensions[i] = (int)d6[i];
        getParameterData(type, nDim, dimensions);
        break;
    case 7:
        unsigned char d7[7];
        fread(d7, sizeof(char), nDim, c3dfile);
        for(i= 0; i < nDim; i++) dimensions[i] = (int)d7[i];
        getParameterData(type, nDim, dimensions);
        break;
    }
    fread(lDescription, sizeof(char), 1, c3dfile);        //Einlesen Länge der Parameterbeschreibung
    lDescr = (int) lDescription[0];
    fread(description+1, sizeof(char), lDescr, c3dfile);  //Einlesen Parameterbeschreibung
}

void c3dData::getParameterData (int type, int nDim, int* dims)
{
    int i=0;
    int nSigns = 1;
    switch(type)                                    //Fallunterscheidung nach einzulesendem Datentyp
    {
        case 255:                                   //Character-Daten
            char *cData;
            for (i = 0; i<nDim; i++)
                nSigns *= dims[i];
            cData = (char*)malloc(nSigns+1);
            fread(cData, sizeof(char), nSigns, c3dfile);

            if(subNames)
            {
                cData[nSigns+1]='\0';
                int l = strcspn(cData, " ");
                name = (char*)malloc(l+1);
                for(i=0; i<=l;i++)
                    name[i]=cData[i];
                name[l+1]='\0';
                puts(name);
                inParamSection = false;
            }
            break;

        case 1:                                     //unsigned Character-Daten (8Bit-Zahl)
            unsigned char ucData[2048];
            for (i = 0; i<nDim; i++)
                nSigns *= dims[i];
            fread(&ucData[0], sizeof(char), nSigns, c3dfile);
            break;
        case 2:                                     //Short Integer-Daten (16 Bit)
            short sData[2048];
            for (i = 0; i<nDim; i++)
                nSigns *= dims[i];
            fread(&sData[0], sizeof(short), nSigns, c3dfile);
            break;
        case 4:                                     //Float-Daten
            float fData[2048];
            for (i = 0; i<nDim; i++)
                nSigns *= dims[i];
            fread(&fData[0], sizeof(float), nSigns, c3dfile);
            break;
    }
}


void c3dData::readDataSection(void)
{
    int dataCounter = 0;
    if(flagFP)
    {
            for(int frame = 1; frame <= duration; frame++)
        {
            for(int j=1; j<=numTraj; j++)
            {
                float f3dData[5];
                fread(f3dData, sizeof(float), 4, c3dfile);
                Data.push_back(f3dData[0]);
                Data.push_back(f3dData[1]);
                Data.push_back(f3dData[2]);
            }
            float analogData[100];
            fread(analogData, sizeof(float), numAnalog, c3dfile);
        }
    }
}
```

A.2 Das csmXtra

csmReader.h

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <fstream>
#include <iostream>
#include <vector>

using namespace std;

class csm
{
public:
    //Variablen
    int nFrames;         //Anzahl Frames
    FILE *csmFile;       //csm Datei
    char* date;          //Erstellungsdatum
    char* actor;         //Darsteller
    float fps;           //Frames pro Sekunde
    int nMarkers;        //Anzahl der Marker
    vector<float> data;  //Positionsdaten

    //Methoden
    int open (char*filename);  //Öffnet das csmFile
    void readCSM(void);        //liest den Header des csmFiles
    void getData(char *line);  //liest die $Point-Sektion
    csm();
};


csm::csm()
{
    nFrames = 0;
    fps = 0;
    actor = " ";
    date = " ";
    nMarkers = 0;
}
//Allgemeine Funktionen
void   deleteLeadingSpaces(char* line);  //löscht führende Leerzeichen
char*  extractWord(char* line);   //liest das nächste Wort aus der Zeile, der Rückgabewert ist die
                                  restliche Zeile
double extractDouble(char* line); //liest den nächsten Doublewert aus der Zeile, der Rückgabewert ist
                                  die restliche Zeile
int    extractInt(char* line);    //liest den nächsten LongInt-Wert aus der Zeile, der Rückgabewert
                                  ist die restliche Zeile
```

csmReader.cpp

```
#include "csmReader.h"

int readCSM(void);
void getData(char* line);

int csm::open(char* filename)
{
    csmFile = fopen(filename,"r");

    if(csmFile)
        readCSM(); //Falls die Datei gelesen werden konnte, lies den Inhalt
        fclose(csmFile);
    else return 0;

    return 1;
}


void csm::readCSM(void)
{
    char *line;
    line = (char*)malloc(256);

    do
    {
        fgets(line, 255, csmFile);
        if(strncmp(line,"$Date",5)==0||strncmp(line,"$date",5)==0) //suche Zeile "$Date"
        {
            if(strlen(line)>6)
            {
                extractWord(line);
                date = extractWord(line);
            }
            else
            {
                fgets(line, 255,csmFile);
                date = extractWord(line);
            }
```

```
        }
        else if(strncmp(line,"$Actor",6)==0||strncmp(line,"$actor",6)==0) //suche Zeile "$Actor"
        {
            if(strlen(line)>7)
            {
                extractWord(line);
                actor = extractWord(line);
            }
            else
            {
                fgets(line, 255,csmFile);
                date = extractWord(line);
            }
        }
        else if (strncmp(line,"$Rate",5)==0||strncmp(line,"$rate",5)==0) //suche Zeile "$Rate"
        {
            if(strlen(line)>6)
            {
                extractWord(line);
                fps = extractDouble(line);
            }
            else
            {
                fgets(line, 255,csmFile);
                fps = extractDouble(line);
            }
        }
    }
    while(strncmp(line,"$Order",6)&&strncmp(line,"$order",6));   //suche Zeile "$Order"

    if(strlen(line)<=7)
        fgets(line, 255,csmFile);

    while(strlen(line)>0)
    {
        extractWord(line);
        nMarkers++;
    }

    do  fgets(line, 255, csmFile);
        while(strncmp(line,"$Points",7)&&strncmp(line,"$points",7)); //suche Zeile "$Points"...

    char *dataline;
    dataline = (char*)malloc(2048*sizeof(char));

    int zaehler =0;
    int fn=0;

    while(fgets(dataline, 2047,csmFile)) //..und lies dann Zeilen(Frame-)weise die Daten aus
    {
        getData(dataline);
    }
}

void csm::getData(char *line)
{
    double f;

    strtol(line, &line,0);          //lies die ersten gültigen Zeichen als Long-Wert ein (Zeilennummer)

    while(strlen(line) > 1)
    {
        f = strtod(line,&line);     //lies die ersten gültigen Zeichen als Double-Wert ein

        if(f==0)                    //...wenn du damit keinen Erfolg hattest, liegt ein Dropout vor
        {
            data.push_back(0.0);    //..setze die folgenden X-,Y- und Z-Werte auf 0.0
            data.push_back(0.0);
            data.push_back(0.0);
            extractWord(line);      //...und lösche das Wort aus der Zeile
        }
        else
            data.push_back(f);
    }
    nFrames++;
}

void deleteLeadingSpaces(char* line)
{
    int counter = 0;
    int count = 0;
    char* buffer;
    buffer = line;
    if(!isgraph(line[0]))
        while(isspace(buffer[++counter]));
            for (int i= counter; i<=strlen(buffer); i++)
                line[count++] = buffer[i];
}

char* extractWord(char* line)
{
    int counter = 0;
```

```
    int count = 0;

    char* word;
    char *buffer;
    deleteLeadingSpaces(line);
    buffer = line;
    while(isgraph(buffer[counter]))
        counter++;
    word = (char*)malloc(counter+1);
    for(int i=0; i<=counter; i++)
        word[i] = buffer[i];
    word[i] = '\0';
    while(counter<=strlen(buffer))
        line[count++] = buffer[counter++];
    deleteLeadingSpaces(line);
    return word;
}

double extractDouble(char* line)
{
    deleteLeadingSpaces(line);
  char* buffer;
    double number = 0.0;
    number = strtod(line, &buffer);
    for (int i = 0;i<=strlen(buffer);i++)
        line[i]=buffer[i];
    deleteLeadingSpaces(line);
    return number;
}

int extractInt(char* line)
{
    deleteLeadingSpaces(line);
    long number = 0;
    number = strtol(line, &line, 0);
    deleteLeadingSpaces(line);
    return (int)number;
}
```

A.3 Das AcclaimXtra

Die Funktionen *deleteLeadingSpaces*, *extractWord*, *extractDouble* und *extractInt* aus der Datei asfReader.cpp sind die Selben wie die bereits im csmReader-Xtra verwendeten und werden daher nicht noch einmal mit aufgeführt.

asfReader.h

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <fstream>
#include <iostream>
#include <vector>

using namespace std;

//////////
//clases//
//////////

/**bone**/
class bone
{
public:
    string name;     //BoneName
    float  length;   // die Länge des Bones
    double xDir;     // die X-Richtung im 3D-Raum zu Beginn
    double yDir;     // die Y-Richtung im 3D-Raum zu Beginn
    double zDir;     // die Z-Richtung im 3D-Raum zu Beginn
    double xAxis;    // die Rotation der X-Achse des Bones bezüglich der 3DWelt
    double yAxis;    // die Rotation der Y-Achse des Bones bezüglich der 3DWelt
    double zAxis;    // die Rotation der Z-Achse des Bones bezüglich der 3DWelt
    string axis;     // die Rotationsreihenfolge der Achsen
    bool   xDof;     // gibt an, on x-Rotationen gespeichert wurden
    bool   yDof;     // gibt an, on y-Rotationen gespeichert wurden
    bool   zDof;     // gibt an, on z-Rotationen gespeichert wurden
    int    id;       // die ID des Bones

    vector<double>   xRot;  // die Rotationsdaten für die X-Achse
```

```
    vector<double>   yRot;  // die Rotationsdaten für die Y-Achse
    vector<double>   zRot;  // die Rotationsdaten für die Z-Achse
};


/**branch**/
class branch     //of hierarchy
{
public:
    char* firstWord;           // der Name des ersten Parentbones
    char* lastWord;            // der Name des zuletzt gelesenen Childbones
    vector<string> branches;   //alle Bonenamen in der richtigen Reihenfolge
    vector<int> ids;            // die Zugehörenden ID-Nummern
};


    void   deleteLeadingSpaces(char* line);
    char*  extractWord(char* line);
    double extractDouble(char* line);
    int    extractInt(char* line);
```

Acclaim.h

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <fstream>
#include <iostream>
#include <vector>

#include "asfReader.h"

/**AcclaimFormat**/
class acclaim
{
public:
//////////
//Methods//
//////////
    acclaim();

    int    openAMC(char* AMC, char* ASF);

    /***ASF-stuff***/
    void   readASF(void);
    void   readUnit(vector<string> lines);
    void   readRoot(vector<string> lines);
    void   readBonedata(vector<string> lines);
    void   readHierarchy(vector<string> lines);
    void   createASFData (void);
    /***AMC-stuff***/
    void   readAMC(void);
    int    readFirstBlock();
    void   readBlocks(int nDof);

    /***preparation for Director exchange***/
    void   writeHierarchy(void);
    void   writeDataList(void);
    void   writeLengthsList(void);
    void   writeDofs(void);
  int  nameCompare(void);


    ////////////////////
    //global variables//
    ////////////////////
    FILE          *ASFdata;
    FILE          *AMCdata;
    vector<string>   lines;
    double        mass;
    double        length;
    float         fps;
    int           angle;
    int           nFrames;
    int           nBones;
    int           nBranches;

    bool          fullySpecified;

    vector<int>      indices;
    vector<string>   names;
    vector<int>      ids;
    vector<float> rotData;
    vector<float> lengths;
    vector<bool>  dofs;
    char* axis;
    vector<string> order;
    vector < vector<float> > rootDof;
    vector<bone>  bones;
    vector<branch>   hierarchy;
```

```
};
```

Acclaim.cpp

```
#include "acclaim.h"

acclaim::acclaim()
{
    mass   = 0.0;
    length = 0.0;
    fps = 0.0;
    angle  = 0;
    nFrames = 0;
    nBones = 0;
    nBranches= 0;

    fullySpecified = false;
}

int acclaim::openAMC(char* AMC, char* ASF)
{
    ASFdata = fopen(ASF,"r");
    AMCdata = fopen(AMC,"r");
    if(ASFdata)
    {
        readASF();
        fclose(ASFdata);
    }
    else
        createASFData();
    if(AMCdata)
    {
        readAMC();
        fclose(AMCdata);
    }
    writeHierarchy();
    writeDataList();
    writeLengthsList();

    return 0;
}

void acclaim::createASFData (void)  //if no amc file was specified
{
    /***** unit ASFdata *****/
    mass = 1.0;
    length = 0.45;
    angle = 1;
    nBones = 30;
    nBranches = 5;
    /***** root ASFdata *****/
    char boneNames[30][10]=
{"lhipjoint","lfemur","ltibia","lfoot","ltoes","rhipjoint","rfemur","rtibia","rfoot","rtoes","lowerback"
,"upperback","thorax","lowerneck","upperneck","head","lclavicle","lhumerus","lradius","lwrist","lhand","
lfingers","lthumb","rclavicle","rhumerus","rradius","rwrist","rhand","rfingers","rthumb"};
    double boneLengths[30] =
{2.5156,7.54288,7.71864,2.18429,1.09718,2.50835,7.43366,7.77122,2.2293,1.1178,1.95077,1.96246,2.00867,1.
75762,1.72601,1.79973,3.47956,5.52302,3.58675,1.79337,0.536412,0.432469,0.620942,3.44099,5.8478,3.62717,
1.81359,0.650516,0.524463,0.753027};
    double boneXDirs [30] = {0.625129,0.34202,0.34202,0.0772037,1.53513e-011,-0.621946,-0.34202,-
0.34202,-0.082694,-1.53565e-011,-0.000695208,-0.00268925,-0.000848408,0.00577709,
0.0320945,0.0120552,0.909,1,1,1,1,1,0.707107,-0.89595,-1,-1,-1,-1,-1,-0.707107};
    double boneYDirs [30] = {-0.738222,-0.939693,-0.939693,-0.212115,-4.21834e-011,-0.740623,-0.939693,-
0.939693,-0.2272,-4.21695e-011,0.988391,0.999885,0.998481,0.998999,0.999136,0.999916,0.371533,-4.48963e-
011,-4.48968e-011,-4.48942e-011,-4.4904e-011,-4.48893e-011,-6.34934e-011,0.416672,-48952e-011,-4.48967e-
011,-4.49008e-11,-4.48812e-011,-4.49228e-011,-6.34912e-011};
    double boneZDirs [30] = {0.253461,0,0,0.97419,1,0.254285,0,0,0.970331,1,-0.151929,-
0.0149203,0.0550912,0.0443608,0.0264105,0.00480151,-0.188897,-6.48352e-028,-4.48968e-011,-9.9435e-027,-
1.9887e-026,-3.9774e-026,0.707107,-0.15813,5.57804e-028,-7.936462-027,-9.9435e-027,-1.9887e-026,-
3.9774e-026,0.707107};
    int boneXAxis[30] = {0,0,0,-90,-90,0,0,0,-90,-90,0,0,0,0,0,0,180,180,0,0,0,-90,0,180,180,0,0,0,-
90};
    int boneYAxis[30] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-30,-30,90,90,90,45,0,30,30,-90,-90,-90,-45};
    int boneZAxis[30] = {0,20,20,20,20,0,-20,-20,-20,-20,0,0,0,00,0,0,-90,-90,90,90,0,0,90,90,-90,-90,-
90,0};
    bool boneXDofs[30] = {0,1,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,1,0,1,1,1,0,1,1,0,1,1,1};

    bool boneYDofs[30] = {0,1,0,0,0,0,1,0,0,0,1,1,1,1,1,1,1,1,0,1,0,0,0,1,1,0,1,0,0,0};
    bool boneZDofs[30] = {0,1,0,1,0,0,1,0,1,0,1,1,1,1,1,1,1,1,0,0,1,0,1,1,1,0,0,1,0,1};

    names.resize(names.size()+1);
    names[0]=("root");
    for(int i=0; i<nBones; i++) //write information for every bone
    {
        names.resize(names.size()+1);
        names[i+1] = boneNames[i];
        bones.resize(bones.size()+1);
        bones[i].id = i;
        bones[i].name = boneNames[i];
```

```
        bones[i].length = boneLengths[i];
        bones[i].xDir = boneXDirs[i];
        bones[i].yDir = boneYDirs[i];
        bones[i].zDir = boneZDirs[i];
        bones[i].xAxis = boneXAxis[i];
        bones[i].yAxis = boneYAxis[i];
        bones[i].zAxis = boneZAxis[i];
        bones[i].xDof = boneXDofs[i];
        bones[i].yDof = boneYDofs[i];
        bones[i].zDof = boneZDofs[i];
    }
    /***** hierarchy ASfdata *****/
    string child;
    hierarchy.resize(hierarchy.size()+1);//first branch: root to ltoes
    child = "root";
    hierarchy[0].branches.push_back(child);
    child = "lhipjoint";
    hierarchy[0].branches.push_back(child);
    child = "lfemur";
    hierarchy[0].branches.push_back(child);
    child = "ltibia";
    hierarchy[0].branches.push_back(child);
    child = "lfoot";
    hierarchy[0].branches.push_back(child);
    child = "ltoes";
    hierarchy[0].branches.push_back(child);

    hierarchy.resize(hierarchy.size()+1);//second branch: root to rtoes
    child = "root";
    hierarchy[1].branches.push_back(child);
    child = "rhipjoint";
    hierarchy[1].branches.push_back(child);
    child = "rfemur";
    hierarchy[1].branches.push_back(child);
    child = "rtibia";
    hierarchy[1].branches.push_back(child);
    child = "rfoot";
    hierarchy[1].branches.push_back(child);
    child = "rtoes";
    hierarchy[1].branches.push_back(child);

    hierarchy.resize(hierarchy.size()+1);//third branch: root to head
    child = "root";
    hierarchy[2].branches.push_back(child);
    child = "lowerback";
    hierarchy[2].branches.push_back(child);
    child = "upperback";
    hierarchy[2].branches.push_back(child);
    child = "thorax";
    hierarchy[2].branches.push_back(child);
    child = "lowerneck";
    hierarchy[2].branches.push_back(child);
    child = "upperneck";
    hierarchy[2].branches.push_back(child);
    child = "head";
    hierarchy[2].branches.push_back(child);

    hierarchy.resize(hierarchy.size()+1);//forth branch: thorax to lfingers
    child = "thorax";
    hierarchy[3].branches.push_back(child);
    child = "lclavicle";
    hierarchy[3].branches.push_back(child);
    child = "lhumerus";
    hierarchy[3].branches.push_back(child);
    child = "lradius";
    hierarchy[3].branches.push_back(child);
    child = "lwrist";
    hierarchy[3].branches.push_back(child);
    child = "lhand";
    hierarchy[3].branches.push_back(child);
    child = "lfingers";
    hierarchy[3].branches.push_back(child);

    hierarchy.resize(hierarchy.size()+1);//fifth branch: thorax to rfingers
    child = "thorax";
    hierarchy[4].branches.push_back(child);
    child = "rclavicle";
    hierarchy[4].branches.push_back(child);
    child = "rhumerus";
    hierarchy[4].branches.push_back(child);
    child = "rradius";
    hierarchy[4].branches.push_back(child);
    child = "rwrist";
    hierarchy[4].branches.push_back(child);
    child = "rhand";
    hierarchy[4].branches.push_back(child);
    child = "rfingers";
    hierarchy[4].branches.push_back(child);

    hierarchy.resize(hierarchy.size()+1);//sixth branch: lwrist to lthumb
    child = "lwrist";
    hierarchy[5].branches.push_back(child);
```

```
    child = "lthumb";
    hierarchy[5].branches.push_back(child);

    hierarchy.resize(hierarchy.size()+1);//sixth branch: rwrist to rthumb
    child = "rwrist";
    hierarchy[6].branches.push_back(child);
    child = "rthumb";
    hierarchy[6].branches.push_back(child);

}

void acclaim::readASF()
{
    char *line;
    line = (char*)malloc(256);

    /***** unit ASFdata *****/
    do fgets(line, 255, ASFdata);        //go to :unit
    while(strncmp(line,":units",6));
    do                              //fill lines with :unit stuff
    {
        fgets(line, 255, ASFdata);
        lines.push_back(line);
    }
    while(strncmp(line,":documentation",14));
    readUnit(lines);
    lines.clear();

    /***** root ASFdata *****/
    do fgets(line, 255, ASFdata);        //go to :root
    while(strncmp(line,":root",5));

    do                              //fill lines with root stuff
    {
        fgets(line, 255, ASFdata);
        lines.push_back(line);
    }
    while(strncmp(line," begin",7));  //go to first bone
    readRoot(lines);
    lines.clear();

    /***** bone ASFdata *****/
    do                              //fill lines with bonedata
    {
        do                          //go through all bones
        {
            fgets(line, 255, ASFdata);
            deleteLeadingSpaces(line);
            lines.push_back(line);
        }
        while(strncmp(line,"end", 3));

        readBonedata(lines);
        lines.clear();
        fgets(line, 255, ASFdata);
    }
    while(strncmp(line,":hierarchy",10));

    /***** hierarchy ASFdata *****/
    do                              //fill lines with hierarchy stuff
    {
        fgets(line, 255, ASFdata);
        lines.push_back(line);
    }
    while(strncmp(line," end",5));
    readHierarchy(lines);
    lines.clear();
}

void acclaim::readUnit (vector<string> lines)
{
    char *word;
    char *buffer;
    for(int i=0; i<lines.size()-1; i++)
    {
        buffer = (char*)malloc(lines[i].size());
        buffer = (char*)lines[i].c_str();
        deleteLeadingSpaces(buffer);
        word = extractWord(buffer);
        deleteLeadingSpaces(buffer);

        if(strncmp(word,"mass",4) == 0)
            mass = extractDouble(buffer);
        else if(strncmp(word,"length",6) == 0)
            length = extractDouble(buffer);
        else if(strncmp(word,"angle",5) == 0)
        {
            char* Angle = extractWord(buffer);
            if(strncmp(Angle, "deg",3) == 0)
                angle = 1;
            else if(strncmp(Angle, "rad",3) == 0)
                angle = 2;
```

```
        }
    }
}

void acclaim::readRoot (vector<string> lines)
{
    char *word;
    char *buffer;
    char* output;

    axis = (char*)malloc(4);

    for(int i=0; i<2; i++)
    {

        output = (char*)malloc(3);

        buffer = (char*)malloc(lines[i].size());
        buffer = (char*)lines[i].c_str();

        deleteLeadingSpaces(buffer);
        word = extractWord(buffer);
        deleteLeadingSpaces(buffer);

        if(strncmp(word, "order", 5) == 0)
        {
            for(int j=0; j<6; j++)
            {
                output = extractWord(buffer);
                order.push_back(output);
                deleteLeadingSpaces(buffer);
            }
        }
        else if(strncmp(word, "axis", 4) == 0)
            axis = extractWord(buffer);
    }
    string root = "root";
    root+='\n';
    names.push_back(root);
}

void acclaim::readBonedata(vector<string> lines)
{

    bones.resize(bones.size()+1);
    bones[nBones].id = nBones;
    bones[nBones].xDof=false;
    bones[nBones].yDof=false;
    bones[nBones].zDof=false;

    char *word;
    char *buffer;
    char *output;

    for(int i=0; i<lines.size()-1; i++)
    {
        buffer = (char*)malloc(lines[i].size());
        buffer = (char*)lines[i].c_str();
        word = extractWord(buffer);

        if(strncmp(word,"name",4)==0)
        {
            deleteLeadingSpaces(buffer);
            output = extractWord(buffer);
            bones[nBones].name = output;
            names.push_back(output);
        }
        else if(strncmp(word,"direction",9)==0)
        {
            bones[nBones].xDir = extractDouble(buffer);
            bones[nBones].yDir = extractDouble(buffer);
            bones[nBones].zDir = extractDouble(buffer);
        }
        else if(strncmp(word,"length",6)==0)
        {
            bones[nBones].length = extractDouble(buffer);
            deleteLeadingSpaces(buffer);
        }
        else if(strncmp(word, "axis",4)==0)
        {
            bones[nBones].xAxis = extractDouble(buffer);
            bones[nBones].yAxis = extractDouble(buffer);
            bones[nBones].zAxis = extractDouble(buffer);
            deleteLeadingSpaces(buffer);
            output = extractWord(buffer);
            bones[nBones].axis = output;
        }


        else if(strncmp(word, "dof",3)==0)
        {
            deleteLeadingSpaces(buffer);
```

```
            output = extractWord(buffer);

            if(strncmp(output,"rx",2)==0)
            {
                bones[nBones].xDof = true;
                if(strlen(buffer)>0)
                {
                    deleteLeadingSpaces(buffer);
                    output = extractWord(buffer);
                    if(strncmp(output,"ry",2)==0)
                    {
                        bones[nBones].yDof = true;
                        if(strlen(buffer)>0)
                        {
                            deleteLeadingSpaces(buffer);
                            output = extractWord(buffer);
                            if(strncmp(output,"rz",2)==0)
                                bones[nBones].zDof = true;
                        }
                    }
                    else if (strncmp(output,"rz",2)==0)
                        bones[nBones].zDof = true;
                }
            }
            else if(strncmp(output,"ry",2)==0)
            {
                bones[nBones].yDof = true;
                if(strlen(buffer)>0)
                {
                    deleteLeadingSpaces(buffer);
                    output = extractWord(buffer);
                    if(strncmp(output,"rz",2)==0)
                        bones[nBones].zDof = true;
                }
            }
            else if(strncmp(output,"rz",2)==0)
                    bones[nBones].zDof = true;
        }

    }
    nBones++;

}

void acclaim::readHierarchy(vector<string> lines)
{
    char* buffer;
    char* firstWord;
    char* word;

    for(int i=1; i<lines.size()-1; i++) //first line (lines[0]) contents "begin"
    {
        buffer = (char*)malloc(lines[i].size());
        buffer = (char*)lines[i].c_str();
        deleteLeadingSpaces(buffer);
        firstWord = extractWord(buffer);
        deleteLeadingSpaces(buffer);

        if(strncmp(firstWord, "root",4)==0)
        {
            while(strlen(buffer)>0)
            {
                word = extractWord(buffer);
                hierarchy.resize(hierarchy.size()+1);
                hierarchy[nBranches].firstWord = firstWord;
                hierarchy[nBranches].lastWord = word;
                hierarchy[nBranches].branches.push_back(firstWord);
                hierarchy[nBranches].branches.push_back(word);
                deleteLeadingSpaces(buffer);
                nBranches++;
            }

        }

        else
        {
            for (int i=0; i < hierarchy.size();i++)
            {
                if(strncmp(hierarchy[i].lastWord, firstWord,strlen(firstWord)-1)==0)
                {
                    word = extractWord(buffer);
                    hierarchy[i].lastWord = word;
                    hierarchy[i].branches.push_back(word);
                    deleteLeadingSpaces(buffer);
                }
            }


            while(strlen(buffer)>1)
            {
                word = extractWord(buffer);
                hierarchy.resize(hierarchy.size()+1);
```

```
                hierarchy[nBranches].firstWord = firstWord;
                hierarchy[nBranches].lastWord = word;
                hierarchy[nBranches].branches.push_back(firstWord);
                            hierarchy[nBranches].branches.push_back(word);
                deleteLeadingSpaces(buffer);
                nBranches++;
            }

        }
    }
}

void acclaim::readAMC(void)
{
    char *line;
    line = (char*)malloc(256);

    do
    {
        fgets(line, 255, AMCdata);
        if(strncmp(line,":FULLY-SPECIFIED",16)==0)
            fullySpecified = true;
        if(strncmp(line,":SAMPLES-PER-SECOND",19)==0)
        {
            extractWord(line);
            fps = extractDouble(line);
        }
    }
    while(strncmp(line,"1",1));
    if(fullySpecified)
    {
        int nDof = readFirstBlock();
        nFrames++;
        readBlocks(nDof);
    }
}

int acclaim::readFirstBlock()
{
    char *line;
    line = (char*)malloc(256);
    char *name;
    int nDof = 0;

    fgets(line, 255, AMCdata);
    extractWord(line);
    deleteLeadingSpaces(line);

    rootDof.resize(6);
    for(int i=0; i<6; i++)
    {
        rootDof[i].push_back(extractDouble(line));
        deleteLeadingSpaces(line);
    }

    while(strncmp(fgets(line, 255, AMCdata), "2", 1))
    {
        name = extractWord(line);
        for(int j=0;j<bones.size(); j++)
        {
            char *bname = (char*)malloc(bones[j].name.size());
            bname = (char*)bones[j].name.c_str();
            if(strncmp(bname, name, strlen(bname)-1) == 0)
                indices.push_back(j);
        }
        if(bones[indices[nDof]].xDof)
            bones[indices[nDof]].xRot.push_back(extractDouble(line));
        if(bones[indices[nDof]].yDof)
            bones[indices[nDof]].yRot.push_back(extractDouble(line));
        if(bones[indices[nDof]].zDof)
            bones[indices[nDof]].zRot.push_back(extractDouble(line));
        nDof++;
    }
    return nDof;
}

void acclaim::readBlocks(int nDof)
{
    char *line;
    line = (char*)malloc(256);

    while(fgets(line, 255, AMCdata))
    {
        extractWord(line);
        deleteLeadingSpaces(line);
        rootDof.resize(6);
        for(int i=0; i<6; i++)
        {
            rootDof[i].push_back(extractDouble(line));
            deleteLeadingSpaces(line);
        }
```

```
        for(i=0; i< nDof; i++)
        {
            fgets(line, 255, AMCdata);
            extractWord(line);
            deleteLeadingSpaces(line);
            if(bones[indices[i]].xDof)
            {
                bones[indices[i]].xRot.push_back(extractDouble(line));
                deleteLeadingSpaces(line);
            }
            else
                bones[indices[i]].xRot.push_back(0.0);
            if(bones[indices[i]].yDof)
            {
                bones[indices[i]].yRot.push_back(extractDouble(line));
                deleteLeadingSpaces(line);
            }
            else
                bones[indices[i]].yRot.push_back(0.0);
            if(bones[indices[i]].zDof)
                bones[indices[i]].zRot.push_back(extractDouble(line));
            else
                bones[indices[i]].zRot.push_back(0.0);
        }

        nFrames++;
        fgets(line, 255, AMCdata);
    }
}

void acclaim::writeHierarchy()
{
    char *word;
    // durchlaufe die ganze Hierarchie
    for(int i=0; i<hierarchy.size(); i++)
    {
        // durchlaufe die einzelnen Äste der Hierarchie
        for(int j=0; j<hierarchy[i].branches.size(); j++)
        {
            // merke dir den Eintrag an dieser Stelle als normalen sting
            word = (char*)hierarchy[i].branches[j].c_str();
            // Vergleiche den Eintrag mit der Liste der Knochennamen, bei Übereinstimmung, merke dir die
KnochenID
            for(int k=0; k<names.size(); k++)
                if(strncmp(word, names[k].c_str(), names[k].size()-1)==0)
                    ids.push_back(k);
        }
        // ist das Ende des Zweiges erreicht, trage -1 ein
        ids.push_back(-1);
    }
}

void acclaim::writeDataList(void)
{
    for(int i=0; i<nFrames-1; i++)
    {
        for(int j=0; j<6;j++)
            rotData.push_back(rootDof[j][i]);

        for( j=0; j<=bones.size();j++)
        {
            if(bones[j].xRot.empty())
            {
                rotData.push_back(0.0);
                rotData.push_back(0.0);
                rotData.push_back(0.0);
            }
            else
            {
                rotData.push_back(bones[j].xRot[i]);
                rotData.push_back(bones[j].yRot[i]);
                rotData.push_back(bones[j].zRot[i]);
            }
        }
    }
}

void acclaim::writeLengthsList(void)
{
    lengths.push_back(0.0);
    for(int i=0; i<=nBones; i++)
        lengths.push_back(bones[i].length);
}

int acclaim::nameCompare (void)
{
    int sum = 0;
    for(int i=0; i<amcNames.size(); i++)
    {
        char *bname = (char*)malloc(amcNames[i].size());
            bname = (char*)amcNames[i].c_str();
        for(int j=0; j <names.size(); j++)
```

```
        {
            char *name = (char*)malloc(names[j].size());
            name = (char*)names[j].c_str();
            if (strncmp(bname,name, names[j].size()-1)==0)
                sum++;
        }
    }
    if (sum == amcNames.size())
        return 1;

    else
        return 0;
}
```

Anhang B: Ausgewählte Lingo-Skripte

B.1 Die globalen Variablen

```
global fObj     -- FileXtra4 object
global c3dData  -- c3dReader object
global csmData  -- csmReader object
global amcData  -- amcReaderObject
global i3d     -- fileIO object
global muiObj    -- MUI Xtra object
-- lists of found files, the files pathnames, i3dfiles and filetypes
global fileList, preList, i3dList, typesList

-- 3d points og big stage
global Points
-- 3D points of small 3d stages
global points1, points2, points3, points4, pointsList
-- Markerdata of big 3d stage
global Markers
-- Markerdata of small 3d stages at beginning
global markers1, markers2, markers3, markers4, markersList

global fNum   -- actual filenumber of big 3d stage

global drive   -- selected directory
global searchDir -- indicates if whole directory should be searched
global pathn   -- holdes pathname of file to be opened
global over   -- indicates wether or not the mouse is over fileselection-textfiels
global actual  -- holds index of actual used listmember

global stoped  -- indicates if scene in big 3d stage is stoped
global looped  -- indicates if scene in big 3d should be looped
global fromList -- indicates if big stage was entered from list or preview window
global edit    -- indicates weather or not the i3d file should be edited
global noI3d   -- indicates if the i3d file already exists

global c3d, amc, csm    -- flags for selected filetypes
global isC3D, isCSM, isASF -- indicates weather c3d, csm or asf file is displayed in big stage
global selectedStage    -- holds Index of selected Stage
global objList, fList, durList, fnList, tList, fpsList -- required lists for small 3d stages

global i3dBuffer  -- holdes i3d content while editing in case of editing is canceld
global BonesList, Root, RotList,dummys -- variables needed for amc display
global asfList, asfPathList, amcFName, asfName
```

B.2 Movie-Skripte

```
------------------
-- SEARCH DRIVE --
------------------
on searchDrive me
 -- list every object directory
 sprite(3).visibility = TRUE
 folders = fObj.fx_FolderToList(drive)

 fileList.deleteAll()
 preList.deleteAll()
 typesList.deleteAll()
 i3dList.deleteAll()
 member("files").text = ("")
 member("messages").text=("")

 if (NOT searchDir) then
  alertInitList=[#buttons:#YesNo,#message:"Include Subfolders?",#icon:#question]
  answer = alert(muiObj, alertInitList)
 else
  answer = 1
 end if

 repeat with i in folders
  -- if c3d-file was found save its name and filepath
  if c3d then
   if fObj.fx_FileGetType(drive&i) = ".c3d" then
    append fileList,(i)
    append preList, (drive)
    info = chars(i, 1, (the number of chars in i -4))
    append i3dList, info & ".i3d"
    append typesList, "c3d"
   end if
  end if
  -- if csm-file was found save its name and filepath
  if csm then
   if fObj.fx_FileGetType(drive&i) = ".csm" then
    append fileList,(i)
    append preList, (drive)
    info = chars(i, 1, (the number of chars in i -4))
    append i3dList, info & ".i3d"
```

```
    append typesList, "csm"
   end if
  end if
  -- if amc-file was found save its name and filepath
  if amc then
   if fObj.fx_FileGetType(drive&i) = ".amc" then
    append fileList,(i)
    append preList, (drive)
    info = chars(i, 1, (the number of chars in i -4))
    append i3dList, info & ".i3d"
    append typesList, "amc"
   end if
   if fObj.fx_FileGetType(drive&i) = ".asf" then
    append asfList,(i)
    append asfPathList, (drive)
   end if
  end if

  -- if also subfolders should be searched
  if answer = 1 then
   -- if subfolder was found, search it
   if i contains "\" then
    pathn = drive&i
    searchSub(i)
   end if
  end if
 end repeat
 member("searching").text = ""
 sprite(12).visibility = TRUE
 writeField()

 if searchDir = TRUE then
  member("messages").text=fileList.count()&" files found on drive "&drive
 else
  member("messages").text=fileList.count()&" files found in "&drive
 end if

end

------------------------
--- CREATE AMC LIST ---
------------------------
on createAmcList (theStage, theObj)
 member(theStage).resetWorld()
 nBones = theObj.amcNBones()
 d = theObj.amcData()

 dur = theObj.amcDuration()

 theList = []
 zaehler = 1
 RootPos = [#XR:[] ,#YR:[] ,#ZR:[]]
 theList.append(RootPos)
 RootRot = [#XR:[] ,#YR:[] ,#ZR:[]]
 theList.append(RootRot)
 BoneRot =[#XR:[] ,#YR:[] ,#ZR:[]]
 theList.append(BoneRot)

 repeat with k = 1 to nBones
  append(theList, BoneRot.duplicate())
 end repeat
 repeat with j = 1 to dur -1
  repeat with i in theList
   append i.XR, (d.getAt(zaehler))
   zaehler = zaehler+1
   append i.YR, (d.getAt(zaehler))
   zaehler = zaehler+1
   append i.ZR, (d.getAt(zaehler))
   zaehler = zaehler+1
  end repeat
 end repeat

 return theList
end



-------------------------------------------
---------  CREATE BONES  ------------------
-------------------------------------------
on createBones (theStage, theObj)
 BResList = []
  m = 30/theObj.asfUnit()
 -- create shader for bones
 bShdr = member(theStage).newShader("s2",#standard)
 bTx = member(theStage).newTexture("tx2",#fromCastMember, member("markerTex"))
 member(theStage).shader("s2").textureList[1] = bTx


 dRes = member(theStage).newModelResource("rdummy",#sphere)
 dRes.radius = 12

 -- create Markers resource
 repeat with i = 1 to theObj.amcNBones()
  append(BResList, member(theStage).newModelResource("RBone"&i,#box))
  BResList[i].length = 20
  BResList[i].width = 20
  BResList[i].height = theObj.asfLength(i-1)*m
  BResList[i].lengthVertices = 2
  BResList[i].widthVertices = 2
  BResList[i].heightVertices = 2

  str = theObj.asfName(i-1)
```

```
  if (str.word[1] = "rhipjoint") or \
    (str.word[1] = "lhipjoint") or \
    (str.word[1] = "rclavicle") or \
    (str.word[1] = "lclavicle") then

   BResList[i].width = theObj.asfLength(i-1)*m
   BResList[i].height = 20
  end if

 end repeat

 -- create Bones
 repeat with i= 1 to theObj.amcNBones()
  member(theStage).newModel("Bone"&i, BResList[i])
  member(theStage).newModel("dummy"&i, dRes)
  thisModel = member(theStage).model("Bone"&i)
  thisDummy = member(theStage).model("dummy"&i)
  member(theStage).model("dummy"&i).shaderList = bShdr
  vec =vector(0,(theObj.asfLength(i-1)*m)/2,0)

  str = theObj.asfName(i-1)
  if (str.word[1] = "lhipjoint") or (str.word[1] = "lclavicle") then

   vec = vector(-(theObj.asfLength(i-1)*m)/2,0,0)
  else if (str.word[1] = "rhipjoint") or (str.word[1] = "rclavicle") then
   vec =vector((theObj.asfLength(i-1)*m)/2,0,0)

  else if (str.word[1] = "lowerback")  or \
  (str.word[1] = "upperback") or \
    (str.word[1] = "thorax")  or \
    (str.word[1] = "lowerneck") or \
    (str.word[1] = "upperneck") or \
    (str.word[1] = "head") then

   vec =vector(0,-(theObj.asfLength(i-1)*m)/2,0)
  end if

  MovePivotPoint(thisModel, vec)
  rotateAxis (thisDummy, thisModel, theObj, i-1)
  thisModel.shaderList = bShdr
 end repeat

 -- create Root
 RRes = member(theStage).newModelResource("RRoot",#sphere)
 RRes.radius = 1
 rootBone = member(theStage).newModel("Root",RRes)


 -- create lights
 aLight = member(theStage).newLight("aL1",#ambient)
 aLight.color = rgb(250,250,250)
 dLight = member(theStage).newLight("dL1",#directional)
 dLight.color = rgb(255,255,255)
 dLight.parent=member(theStage).camera("defaultView")
 dLight.worldPosition = member(theStage).camera("defaultView").worldPosition

 -- create Floor with shader and texture
 PlRes = member(theStage).newModelResource("RFloor",#plane)
 PlRes.length = 10000
 PlRes.width = 10000

 fShdr = member(theStage).newShader("s1",#standard)
 fShdr.specular = rgb( 0, 0, 0 )
 fTx = member(theStage).newTexture("tx1",#fromCastMember, member("floorTex"))
 fTx.quality=(#high)
 fShdr.textureList[1] = fTx

 member(theStage).newModel("Floor", PlRes)
 member(theStage).model("Floor").shaderList = fShdr
 member(theStage).model("Floor").transform.rotation = vector(90,0,0)

 -- create Origin
 orig = member(theStage).newModel("RotPoint",RRes)
end

-----------------------------------------------
---------  MOVE PIVOT POINT  -------------
-----------------------------------------------
on MovePivotPoint (thisModel, thisVector)

 if not(findPos(thisModel.modifier,#meshDeform))then
  thisModel.addModifier(#meshDeform)
 end if

 repeat with meshIndex = 1 to thisModel.meshDeform.mesh.count

  thisVertexList = thisModel.meshDeform.mesh[meshIndex].vertexList

  repeat with vertexIndex = 1 to thisVertexList.count
   thisVertex = thisVertexList[vertexIndex]
   thisVertex = thisVertex + thisVector

   thisVertexList[vertexIndex] = thisVertex
  end repeat
  thisModel.meshDeform.mesh[meshIndex].vertexList = thisVertexList
 end repeat
end MovePivotPoint

-----------------------------------------------
------- BUILD HIERARCHY -------------------
-----------------------------------------------
on BuildHierarchy (theStage, theObj)
 hier = theObj.asfHierarchy()
 counter = 1
```

```
 repeat while (counter < hier.count)
  repeat while NOT(hier[counter]=-1)
   parentBone = hier[counter]
   childBone = hier[counter +1]
   if (parentBone = 0) then
    theParent = member(theStage).model("Root")

   else
    theParent = member(theStage).model("Bone"&parentBone)

   end if
   l = theObj.asfLength(childBone-1)*30/theObj.asfUnit()
   if NOT(childBone = -1) then
    theChild = member(theStage).model("dummy"&childBone)
    startPos = theParent.worldposition
    dir = theObj.amcDirections(childBone-1)
    vec = vector(dir[1], dir[2], dir[3])

    bonePos = startPos + vec*l

    theChild.transform.position = bonePos
    theParent.addChild(theChild,#preserveWorld)

   end if
   counter = counter + 1
  end repeat
  counter = counter + 1
 end repeat
end
-----------------------
---  ROTATE AXIS  ---
-----------------------
on rotateAxis (thisDummy, thisModel, theObj, i)
 axis = theObj.amcAxis(i)
 a = axis[1]
 b = axis[2]
 c = axis[3]

 thisDummy.addChild(thisModel)
 thisDummy.transform.rotation = vector(a, b, c)
end
```

B.3 Frame-Skripte

```
----------------------------
---  INIT SMALL STAGES  ---
----------------------------
on exitFrame me
 posN = pos
 fList.deleteAll()
 objList.deleteAll()
 durList.deleteAll()
 fnList.deleteAll()
 tList.deleteAll()
 fpsList.deleteAll()

 counter = 1

 if fileList.count() - pos >=4 then
  repeat with i= 1 to 4
   member("name"&counter).text = ""
   fileN = preList[posN]&fileList[posN]
   append fList, fileN
   if(typesList[posN] = "c3d") then
    append objList,new(Xtra "c3dReader")
    append tList, "c3d"
   else if(typesList[posN] = "csm") then
    append objList,new(Xtra "csmReader")
    append tList, "csm"
   else if(typesList[posN] = "amc") then
    append objList,new(Xtra "AmcReader")
    append tList, "amc"
   end if
   member("name"&counter).text = fileList[posN]
   posN = posN+ 1
   counter = counter+1
  end repeat

 else if fileList.count() >= 4 AND fileList.count()-pos<4 then
  repeat with i = pos to fileList.count()
   member("name"&counter).text = ""
   fileN = preList[posN]&fileList[posN]
   append fList, fileN
   if(typesList[posN] = "c3d") then
    append objList,new(Xtra "c3dReader")
    append tList, "c3d"
   else if(typesList[posN] = "csm") then
    append objList,new(Xtra "csmReader")
    append tList, "csm"
   else if(typesList[posN] = "amc") then
    append objList,new(Xtra "AmcReader")
    append tList, "amc"
   end if
   member("name"&counter).text = fileList[posN]
   posN = posN+ 1
   counter = counter+1
  end repeat
 else
  repeat with i = 1 to fileList.count()
```

```
   member("name"&counter).text = ""
   fileN = preList[posN]&fileList[posN]
   append fList, fileN
   if(typesList[posN] = "c3d") then
    append objList,new(Xtra "c3dReader")
    append tList, "c3d"
   else if(typesList[posN] = "csm") then
    append objList,new(Xtra "csmReader")
    append tList, "csm"
   else if(typesList[posN] = "amc") then
    append objList,new(Xtra "AmcReader")
    append tList, "amc"
   end if
   member("name"&counter).text = fileList[posN]
   posN = posN+ 1
   counter = counter+1
  end repeat
 end if

 pointsList.deleteAll()
 markersList.deleteAll()
 repeat with i= 1 to fList.count()
  if(tList[i] = "c3d") then
   objList[i].c3dOpen(fList[i])
   append durList, objList[i].c3dDuration()
   append fpsList, objList[i].c3dFrameRate()
   append markersList, createList("file"&i, objList[i], "c3d")
   append pointsList, createPoints("file"&i, objList[i], "c3d")
   cam = member("file"&i).camera("defaultView")
   cam.transform.position = vector(9.5832, -4764.4077, 1496.2941)
   cam.transform.rotation = vector(84,0,0)
  else if (tList[i] = "csm") then
   objList[i].csmOpen(fList[i])
   append durList, objList[i].csmDuration()
   append fpsList, objList[i].csmRate()
   append markersList, createList("file"&i, objList[i], "csm")
   append pointsList, createPoints("file"&i, objList[i], "csm")
   cam = member("file"&i).camera("defaultView")
   cam.transform.position = vector(9.5832, -4764.4077, 1496.2941)
   cam.transform.rotation = vector(84,0,0)
  else if (tList[i] = "amc") then
   objList[i].amcOpen(fList[i], asfPathList[1]&asfList[1])
   append durList, objList[i].amcDuration()
   append fpsList, objList[i].amcFps()
   append markersList, createAmcList("file"&i, objList[i])
   append pointsList, void
   createBones("file"&i, objList[i])
   buildHierarchy("file"&i, objList[i])
   cam = member("file"&i).camera("defaultView")
   cam.transform.position = vector(-91.4433, 1120.9, 5801.55)
   cam.transform.rotation = vector(-3,0,0)
  end if

  append fnList, 1

 end repeat
end
------------------------
---  LOOP STAGES  ---
------------------------
on exitFrame me
 member("searching").text = ""
 sprite(12).visibility = TRUE
 repeat with i = objList.count() down to 1
  if (tList[i]="c3d") or (tList[i]="csm") then

   counter = 1
   if (fnList[i] < durList[i]-1) then
    repeat with j in markersList[i]     -- go through markersList[i]
     newIndex = fnList[i]
     newPos = vector((j.XP[newIndex]),(j.YP[newIndex]),(j.ZP[newIndex]))
     pointsList[i][counter].transform.position = newPos
     counter = counter +1
    end repeat
   end if

  else if (tList[i] = "amc") then
   fNum = fnList[i]
   dur = durList[i]
   m = 30/objList[i].asfUnit()
   bList = markersList[i]
   if (fnList[i] < durList[i]-1) then
    posVec = vector(bList[1].XR[fNum],bList[1].YR[fNum],bList[1].ZR[fNum])
    rotVec = vector(bList[2].XR[fNum],bList[2].YR[fNum],bList[2].ZR[fNum])
    member("file"&i).model("root").transform.position=posVec*m
    member("file"&i).model("root").transform.rotation=rotVec

    repeat with j=1 to 30
     theBone = member("file"&i).model("Bone"&j)
     theDummy = theBone.parent
     theParent = theDummy.parent

     xr = bList[j+2].XR[fNum]
     yr = bList[j+2].YR[fNum]
     zr = bList[j+2].ZR[fNum]
     rotVec = vector(xr,yr,zr)
     theBone.transform.rotation=( rotVec)

     theDummy.worldPosition = theParent.worldPosition

     vec =vector(0,(objList[i].asfLength(j-1)*m),0)
     str = objList[i].asfName(j-1)
     if (str.word[1] = "lhipjoint") or (str.word[1] = "lclavicle") then
      vec = vector(-(objList[i].asfLength(j-1)*m),0,0)
     else if (str.word[1] = "rhipjoint") or (str.word[1] = "rclavicle") then
```

```
      vec =vector((objList[i].asfLength(j-1)*m),0,0)
     else if (str.word[1] = "lowerback")  or \
         (str.word[1] = "upperback") or \
         (str.word[1] = "thorax")  or \
         (str.word[1] = "lowerneck") or \
         (str.word[1] = "upperneck") or \
         (str.word[1] = "head") then
      vec =vector(0,-(objList[i].asfLength(j-1)*m),0)
     end if

     theBone.translate(-vec,#self)
     bonePos = theBone.worldPosition
     theBone.translate(vec,#self)
     theDummy.worldPosition = bonePos
    end repeat
   end if
  end if


  if (fnList[i] < durList[i]-14)then
   factor = fpsList[i]/12
   setAt fnList, i, fnList[i] + factor
  else
   setAt fnList, i, 1
  end if
 end repeat

 go to the frame
end

property spriteNum, pMousedDown
property m_n, m_hi, m_dn
global edit

on beginSprite me
 pMousedDown = 0
 if edit then
  sprite(me.spritenum).member = member"save"
  m_n = member("save")
  m_hi = member("save_S")
  m_dn = member("save_P")
 else
  sprite(me.spritenum).member = member"save_inactive"
  m_n = member("save_inactive")
  m_hi = member("save_inactive")
  m_dn = member("save_inactive")
 end if
end
```

B.4 Member-Skripte

```
-----------------------
---  EDIT BUTTON  ---
-----------------------
property spriteNum, pMousedDown
property m_n, m_hi, m_dn

on beginSprite me
 pMousedDown = 0
 if noI3d then
  sprite(me.spritenum).member = member("create")
  m_n = member("create")
  m_hi = member("create_S")
  m_dn = member("create_P")
 else
  if edit then
   sprite(me.spritenum).member = member("cancel")
   m_n = member("cancel")
   m_hi = member("cancel_S")
   m_dn = member("cancel_P")
  else
   sprite(me.spritenum).member = member("edit")
   m_n = member("edit")
   m_hi = member("edit_S")
   m_dn = member("edit_P")
  end if
 end if
end

on mouseDown me
 pMousedDown = 1
 sprite(me.spritenum).member = m_dn
end

on mouseUp me
 if NOT(pMousedDown) then exit
 pMousedDown = 0
 sprite(me.spritenum).member = m_hi
 updatestage
 doMyStuff me
end

on mouseUpOutside me
 pMousedDown = 0
 sprite(me.spritenum).member = m_n
end

on mousewithin me
 if not(the stilldown) then
```

```
  sprite(me.spritenum).member = m_hi
 end if
end

on mouseEnter me
 if pMousedDown then
  sprite(me.spritenum).member = m_dn
 else if not (the stilldown) then
  sprite(me.spritenum).member = m_hi
 end if
end

on mouseLeave me
 if NOT(the stilldown) OR pMousedDown then
  sprite(me.spritenum).member = m_n
 end if
end

on doMyStuff me
 fname = preList[actual]&i3dList[actual]
 if noI3d then
  if edit then
   -- change buttons' surfaces
   edit = false
   sprite(me.spritenum).member = "edit"
   m_n = member("create")
   m_hi = member("create_S")
   m_dn = member("create_P")
   sprite(34).member = member "save_inactive"
   sprite(34). m_n = member("save_inactive")
   sprite(34).m_hi = member("save_inactive")
   sprite(34).m_dn = member("save_inactive")
   -- change textfields' state
   repeat with i=23 to 33
    num = i
    editField (num)
   end repeat
   -- clear fields
   member("tfDate").text =   ""
   member("tfGroup").text =   ""
   member("tfSubject").text =  ""
   member("tfDuration").text = ""
   member("tfFramerate").text = ""
   member("tfCleaned").text =  ""
   member("tfLastDate").text = ""
   member("tfModified").text = ""
   member("tfSegment").text =  ""
   member("tfFile").text = ""
   member("tfComment").text =  ""
  else
   -- change buttons' surfaces
   edit = true
   sprite(me.spritenum).member = "cancel"
   m_n = member("cancel")
   m_hi = member("cancel_S")
   m_dn = member("cancel_P")
   sprite(34).member = member "save"
   sprite(34). m_n = member("save")
   sprite(34).m_hi = member("save_S")
   sprite(34).m_dn = member("save_P")
   -- insert known data
   if(isC3D) then
    member("tfDuration").text = string(c3dData.c3dDuration())
    member("tfFramerate").text = string(c3dData.c3dFrameRate())
    member("tfSubject").text  = string(word 1 of c3dData.c3dSubject())
   else if (isCSM) then
    member("tfDuration").text = string(csmData.csmDuration())
    member("tfFramerate").text = string(csmData.csmRate())
    member("tfSubject").text  = string(word 1 of csmData.csmActor())
    member("tfDate").text   = string(word 1 of csmData.csmDate())
   else if (isAMC) then
    member("tfDuration").text = string(amcData.amcDuration())
    member("tfFramerate").text = string(amcData.amcFps())
    member("tfFile").text   = asfName
   end if
   -- change textfields' state
   repeat with i=23 to 32
    num = i
    editField (num)
   end repeat
  end if
 else
  if edit then
   -- change buttons' surfaces
   edit = false
   sprite(me.spritenum).member = "edit"
   m_n = member("edit")
   m_hi = member("edit_S")
   m_dn = member("edit_P")
   sprite(34).member = member "save_inactive"
   sprite(34). m_n = member("save_inactive")
   sprite(34).m_hi = member("save_inactive")
   sprite(34).m_dn = member("save_inactive")
   -- change textfields' state
   repeat with i=23 to 32
    num = i
    editField (num)
   end repeat

   -- restore old i3d content
   member("tfDate").text = i3dBuffer.line[1].word[2..(the number of words in i3dBuffer.line[1])]
   member("tfGroup").text = i3dBuffer.line[2].word[2..(the number of words in i3dBuffer.line[2])]
   member("tfSubject").text = i3dBuffer.line[3].word[2..(the number of words in i3dBuffer.line[3])]
   member("tfDuration").text = i3dBuffer.line[4].word[2..(the number of words in i3dBuffer.line[4])]
   member("tfFramerate").text = i3dBuffer.line[5].word[2..(the number of words in i3dBuffer.line[5])]
```

```
    member("tfCleaned").text = i3dBuffer.line[6].word[2..(the number of words in i3dBuffer.line[6])]
    member("tfLastDate").text = i3dBuffer.line[7].word[4..(the number of words in i3dBuffer.line[7])]
    member("tfModified").text = i3dBuffer.line[8].word[3..(the number of words in i3dBuffer.line[8])]
    member("tfSegment").text = i3dBuffer.line[9].word[3..(the number of words in i3dBuffer.line[9])]
    member("tfFile").text = i3dBuffer.line[10].word[3..(the number of words in i3dBuffer.line[10])]
    member("tfComment").text = i3dBuffer.line[11].word[2..(the number of words in i3dBuffer.line[11])]
   else

    edit = true

    -- change buttons' surfaces
    sprite(me.spritenum).member = "cancel"
    m_n = member("cancel")
    m_hi = member("cancel_S")
    m_dn = member("cancel_P")
    sprite(34).member = member "save"
    sprite(34). m_n = member("save")
    sprite(34).m_hi = member("save_S")
    sprite(34).m_dn = member("save_P")
    -- change textfields' state
    repeat with i=23 to 32
     num = i
     editField (num)
    end repeat
   end if
  end if
end
```

Literatur- und Quellenverzeichnis

Eberl, Markus/Jacobsen, Jens: Macromedia Director 8.5 - Das komplette Wissen für Multimediapublisher, 1. Auflage, München: Markt + Technik Verlag, 2002

Gross, Phil/ Gross, Mike: Macromedia Director 8.5 Shockwave Studio für 3D - Das offizielle Trainingshandbuch 1. Auflage, München: Markt + Technik Verlag, 2002

Tab Julius: A Little Something Xtra, MX Developer's Journal 02/2004

Tab Julius: Architecting with Director, MX Developer's Journal 03/2004

Tab Julius: MOA City, MX Developer's Journal 04/2004

Motion Lab Systems: C3D Format, User Guide, http:\\www.motion-labs.com

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum

Unterschrift